VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Absolvování individuální odborné praxe

Individual Professional Practice in the Company

2020 **David Čapčuch**

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra telekomunikační techniky

Zadání bakalářské práce

Student: **David Čapčuch**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R059 Mobilní technologie

Téma: Absolvování individuální odborné praxe
Individual Professional Practice in the Company

Jazyk vypracování:

Zásady pro vypracování:

1. Student vykoná individuální praxi ve firmě: SCOVECO, s.r.o.
2. Struktura závěrečné zprávy:
a. Popis odborného zaměření firmy, u které student vykonal odbornou praxi a popis pracovního zařazení studenta
b. Seznam úkolů zadaných studentovi v průběhu odborné praxe s vyjádřením jejich časové náročnosti
c. Zvolený postup řešení zadaných úkolů
d. Teoretické a praktické znalosti a dovednosti získané v průběhu studia uplatněné studentem v průběhu odborné praxe
e. Znalosti či dovednosti scházející studentovi v průběhu odborné praxe
f. Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení

Seznam doporučené odborné literatury:

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Zdeňka Chmelíková, Ph.D.**

Konzultant bakalářské práce: Ing. Zdeněk Velart

Datum zadání: 01.09.2019

Datum odevzdání: 30.04.2020

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA * Fakulta elektrotechniky a informatiky * OSTRAVA * 1D16

prof. Ing. Miroslav Vozňák, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

„Prohlašuji, že jsem tuto bakalářskou/diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal."

V Ostravě dne: 12. května 2020 .....

Prohlášení zástupce spolupracující právnické nebo fyzické osoby

„Souhlasím se zveřejněním této bakalářské/diplomové práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v bakalářských/magisterských programech VŠB-TU Ostrava."

Dne: 12. května 2020

.....

podpis zástupce

Poděkování

Chtěl bych tímto poděkovat panu Ing. Zdeňkovi Velartovi, Ph.D. za jeho ochotný a vstřícný přístup, zpětné vazby na kód, zkušenosti, které mi předal a veškeré jeho rady, které mi poskytnul. Dále firmě SCOVECO s.r.o. za poskytnutí možnosti absolvování odborné praxe.

Abstrakt

Tato bakalářská práce popisuje průběh absolvování mé odborné praxe ve firmě SCOVECO s.r.o., ve které jsem působil jako programátor Java.

V práci představím firmu SCOVECO s.r.o., její působnost a zaměření, dále následuji teoretickou a praktickou části. Uvnitř teoretické části jsou popsány a vysvětleny technologie, se kterými jsem se při absolvování praxe setkal a jejich využití. V praktické části jsou uvedeny zadané úkoly a moje řešení včetně postupů při plnění těchto úkolů. Dále uvedu přínos odborné praxe v podobě získaných znalostí a dovedností. Na závěr shrnu dosažené výsledky a zhodnotím odbornou praxi.

Klíčová slova

Metrika, Java, Jakarta, MariaDB, Liferay, Service Builder, Tomcat, JavaServer Faces, PrimeFaces, DBeaver, Git, xhtml, SCOVECO s.r.o.

Abstract

This bachelor thesis describes my experience of individual professional practice in company SCOVECO s.r.o., in which I worked as a Java programmer.

I start by introducing the company SCOVECO s.r.o. regarding its scope of work and focus, then I move on to theoretical and practical sections. In theoretical section I describe and explain the technologies I encountered during the professional practice and their uses. In practical section I list all the tasks assigned to me along with my solutions and approaches to solving them. After that, I list the benefits of the professional practice in the form of knowledge and experience gain. Finally, I summarize achieved results and give an assessment of the professional practice.

Key words

Metric, Java, Jakarta, MariaDB, Liferay, Service Builder, Tomcat, JavaServer Faces, PrimeFaces, DBeaver, Git, xhtml, SCOVECO s.r.o.

Obsah

Seznam použitých symbolů a zkratek

Java EE – Java Enterprise Edition

Jakarta EE – Jakarta Enterprise Edition

JSF – JavaServer Faces

JSP – JavaServer Pages

Java SE – Java Standard Edition

API – Application Programming Interface

HTTP – Hypertext Transfer Protocol

JSON – JavaScript Object Notation

XML – Extensible Markup Language

IDE – Integrated development environment

GUI – Graphical User Interface

PDE – Plug-in Development Environment

JPA – Java Persistence API

DSL – Domain Specific Language

HTML – Hypertext Markup Language

XHTML – Extensible Hypertext Markup Language

XLS – Excel Spreadsheet

PDF – Portable Document Format

CSV – Comma-Separated Values

AJAX – Asynchronous JavaScript and XML

SQL – Structured Query Language

JDBC – Java Database Connectivity

RDS – Relational Database Service

ACID – Atomicity, Consistency, Isolation, Durability

URL – Uniform Resource Locator

ID – Identification

JDK – Java Development Kit

CSS – Cascading Style Sheets

ASPICE – Automotive Software Performance Improvement and Capability Determination

Seznam ilustrací a seznam tabulek

1. ÚVOD

Bakalářskou práci jsem vykonával ve společnosti SCOVECO s.r.o. na pozici Java programátora. Zvolil jsem si právě tuhle odbornou praxi z důvodu, že mě programátorská práce baví a chci se v ní zlepšovat, tudíž šance získat reálné zkušenosti pro mě byla velmi cenná.

Během praxe jsem se setkal s mnoha novými technologiemi, se kterými jsem v minulosti neměl možnost pracovat. Účelem této práce bylo vykonat analýzu, implementaci a testování systému „Metrics“ na zobrazení, vytváření, mazání, úpravu a monitorování metrik pro splnění standardu ASPICE za pomocí použití programovacího jazyka Java, databáze a webového uživatelského rozhraní.

1.1. Popis firmy

Firma SCOVECO byla založena roku 2011 a momentálně sídlí v Ostravě. Cílem firmy je prosazovat jednoduchá a inovativní řešení jednoduchých i komplexních problémů pomocí moderních technologií. Zaměřuje se na vývoj informačních a cloudových systémů, mobilních řešení a analýzu a zpracování různých požadavků. [1]

Po nástupu do firmy SCOVECO s.r.o. mi pán Ing. Zdeněk Velart, Ph.D. vysvětlil, jaké bude mé zaměření práce a které technologie si mám nastudovat. Dále jsem se obeznámil s pracovištěm a byl jsem zařazen s dalším studentem do dvoučlenného týmu.

SCOVECO s.r.o. kancelář se nachází v areálu Vysoké Školy Báňské v budově Podnikatelského inkubátoru, kde jsem po většinu absolvování odborné praxe na vlastním zařízení vykonával práci. Od začátku března bylo vzhledem k tehdejší situaci v České republice jak nežádoucí, tak i nemožné fyzicky pracovat v kanceláři. Díky původu práce jsem měl možnost pracovat z domu, kde se neprojevilo žádné omezení, zvlášť díky verzovacího systému Git, díky kterému bylo snadné sledování změn a kritika kódu.

2. TEORETICKÁ ČÁST

V této části popisuji použité technologie a nástroje, které byly využity v mé bakalářské práci. Ke každé popisuji důvody využití a popis vlastností, které sloužily pro mé pracovní účely při absolvování odborné praxe.

2.1. Jazyk Java

Java je jak vysokoúrovňový (snadno čitelný a zapisovatelný pro člověka) programovací jazyk, tak výpočetní platforma vydána firmou Sun Microsystems v roce 1995. Mnoho aplikací i webů je založeno na Javě a bez nainstalované Java platformy nejsou funkční. Java se používá na běžných počítačích, v datacentrech, herních konzolích, vědeckých superpočítačích, i na mobilech, zkrátka tedy skoro všude, kde je moderní technologie. Java (k dnešnímu datu 10.03.2020) je dostupná zdarma. [2]

Java jako programovací jazyk:

Java byla vytvořena na základě požadavků výpočetních prostředí, na kterých musí být software nasazen. Vysoký růst internetu si vyžádal nový pohled na to, jak se budou vyvíjet a distribuovat aplikace. V dnešním světě elektronické komerce a distribuce se musela Java přizpůsobit různým požadavkům:

- **Robustnost** – jazyk byl navržen tak, aby byl velice spolehlivý. Poskytuje rozsáhlou kontrolu kompilace, i kontrolu za běhu programu. Jazyk navádí programátory k spolehlivým programovacím zvykům, například zde neexistují datové typy pointer a o správu paměti se stará tzv. „sběratel“, což eliminuje mnoho programovacích chyb.
- **Bezpečnost** – vzhledem k tomu, že je Java navržena tak, aby fungovala v distribuovaných prostředích, byla bezpečnost velkým faktorem. Java umožňuje vytvářet aplikace tak, že nemohou být napadeny zvenku. Například v síťovém prostředí je aplikace napsaná v Javě zabezpečena proti škodlivým kódům, které se snaží vniknout do sítě a vytvořit viry, či napadnout důležité soubory.
- **Vysoká výkonnost** – Java dosahuje vysokého výkonu díky použití schématu, které umožňuje interpretovi (program, který čte zdrojový kód řádek po řádku a vykoná dané aktivity na základě instrukcí [3]) běžet na maximum bez toho, aby musel kontrolovat běžící prostředí. Zároveň díky sběrateli je vždy velká pravděpodobnost, že bude dostupná paměť, kdykoliv bude potřeba.
- **Využití více vláken** – podpora využití více vláken je implementována na jazykové úrovni pomocí třídy „Thread“. Tato třída byla napsána tak, že funkcionalita programu je bezkonfliktní a není ovlivněna, pokud je spuštěno více vláken najednou. Díky využití více vláken může vývojář poskytnout uživateli vysokou úroveň interaktivity aplikace.
- **Podpora více platform** – v různých prostředí může být různý jak hardware, tak software. Je důležité, aby program napsaný pro určité zařízení správně běžel i na jiném zařízení. Java kompilátor ze zdrojového kódu, který je zapsaný v prostém textu s příponou .java generuje bytecode s příponou .class, který není pro procesor přirozený, místo toho se spouští na „Java Virtual Machine“, který je součástí Java platformy. Bytecode je tedy formát nezávislý na architektuře navržen tak, aby efektivně přenesl kód na více hardware a software platformách.

- **Přenositelnost** – Díky specifikaci „Java Virtual Machine“ Java zaručuje, že chování (například funkce aritmetických operátorů) a velikosti datových typů na jedné platformě zůstane stejné, když se program přenese na jinou platformu, protože vygenerovaný bytecode vždy bude interpretován stejně.

Zároveň byl kladen důraz na jednoduchost a obeznámenost (jazyk je podobný C++) tak, aby se vývojáři mohli jazyk snadno naučit. Z těchto důvodů, a z důvodu využití moderních softwarových vyvíjecích metodik byl jazyk navržen jako objektově orientovaný. [4]

Java jako platforma:

Java platforma se od ostatních platform liší tím, že je čistě softwarová, která běží na ostatních, hardwarových platformách. Skládá se ze dvou částí:

- **Java Virtual Machine** – interpretuje bytecode generovány kompilátorem.
- **Java API** – poskytuje hlavní jádro funkcionality Java programovacího jazyku. Nabízí širokou škálu užitečných tříd připravených k použití v aplikacích. Zahrnuje vše od základních objektů, přes síť a zabezpečení, až po generování XML, přístup k databázi a další. [5]

2.2. Platforma Jakarta Enterprise Edition

Dříve známa jako Java EE, je rozšíření Java SE platformy. Platforma Jakarta EE (dále už jen Jakarta) poskytuje API (kolekce softwarových komponentů, které se používají k vytváření dalších komponentů, či aplikací) a běhové prostředí pro vývoj a běh rozsáhlých, vícevrstvých, škálovatelných, spolehlivých a bezpečných síťových aplikací. [6] Narozdíl od Java SE poskytuje specifikace pro podnikové funkce, jako například distribuované výpočetní techniky a webové služby. Jakarta aplikace jsou typu server-side (aplikace běží na serveru) a běží většinou na mikro serverech, nebo aplikačních serverech. Možné využití je například v elektronické komerci, účetnictví a bankovních informačních systémech. [7]

Cíle Jakarty

Architektura Jakarty zjednodušuje nejběžnější problémy, které vývojáři musí při tvoření moderních aplikací řešit a umožňuje snadnější použití známých návrhových vzorů a osvědčených postupů. Různé příklady častých problémů zahrnují:

- zpracování žádostí, které přicházejí od klienta z webového prohlížeče. Jako řešení tohoto problému poskytuje JSP API, které obsahují metody pro zpracování různých událostí, jako například zjištění, co uživatel napsal do textového pole, nebo uložení HTTP cookie (malý textový soubor, který slouží k rozlišení uživatelů a obsahuje informace o tom, jak daní uživatelé využívají stránku [8]) do uživatelského prohlížeče.
- Ukládání a načítání informaci z databáze. Pro řešení tohoto problému Jakarta poskytuje Java Persistence API, díky kterému se snadno mapují data používány uvnitř programu na data uvnitř tabulek v databázi. [9]

Hlavní specifikace Jakarty

1. **Webové specifikace:**
   - Servlet – definuje správu HTTP žádostí.
   - WebSocket – komunikační protokol, který zajišťuje komunikaci mezi klientem a serverem.
   - JSP – pomáhá vytvářet GUI a zpracovávat vstup klienta.
   - Unified Expression Language – jednoduchý jazyk umožňující přístup ke komponentům JSP
2. **Specifikace webových služeb:**
   - API pro zpracování JSONu – sada pro zpracování informací v JSON formátu. (jednoduchý, snadno čitelný a zapisovatelný formát pro ukládání a transportování dat [10])
   - API pro vázání JSONu – sada pro vázání a parsování JSON souboru do Java třídy.
   - Java architektura pro vázání XML – umožňuje vázání XML do Java objektů. (XML – značkový jazyk se zaměřením na nesení dat ve formátu prostého textu, díky kterému umožňuje tyto data uložit, přenést a sdílet bez ohledu na hardware a software [11])
3. **Podnikové specifikace:**
   - Kontexty a závislosti – podporuje používání závislostí v projektu
   - Enterprise JavaBean – je sada jednoduchých API, které poskytují transakce, vzdálené volání procedur (spuštění procedury v jiném prostředí, než od klienta a následný zpětný přenos výsledků procedury ke klientovi [12]) a kontrola souběžnosti.
   - Persistence API – mapování objektů mezi relačními databázemi a Java třídami.
   - Transaction API – poskytuje rozhraní a anotace pro podporu transakcí. [7]

2.3. Vývojové prostředí Eclipse IDE

Je zdarma ke stažení desktopové vývojové prostředí s otevřeným kódem, [13] určeno pro vývoj v různých programovacích jazycích, nejvíce známé pro podporu jazyka Java, konkrétně Javy SE a Jakarty. Mezi další jazyky, které prostředí podporuje patří: C, C++, JavaScript, TypeScript, a PHP. Je ve vývoji od roku 2001. Hlavní vlastnosti Eclipse IDE je široká škála kolekcí nástrojů, které si uživatel může nainstalovat, například pomocníky pro tvorbu GUI, nástroje pro modelování, mapování, hlášení a testování. Dále je zde možnost využití Tržiště, kde je možnost hledat, instalovat a hlasovat pro nové zásuvné moduly, kterých je momentálně dostupných tisíce a díky kterým se dá přizpůsobovat prostředí Eclipse IDE podle vlastních představ. Tyto zásuvné moduly si mohou uživatelé vytvářet a publikovat i sami pomocí PDE. [14] Mezi nejznámější zásuvné moduly patří „Darkest Dark Theme with DevStyle“, neboli tmavý námět prostředí a SonarLint, který dokáže detekovat a opravit problémy s kvalitou. [15]

Mezi dvě nejznámější distribuce Eclipse IDE patří:

- **Eclipse IDE for Enterprise Java Developers**, která obsahuje podporu Jakarty, tvoření webových aplikací, prostředí pro Javu, JPA, JSF, Mylyn, Maven, Gradle a klienta Git.
- **Eclipse IDE for Java Developers**, která obsahuje podporu pro Java SE, prostředí pro Javu, klienta Git, editor pro XML, Mylyn, Maven a Gradle. [16]

2.4. Budovací nástroj Gradle

Gradle je nástroj s otevřeným kódem pro budování programu neboli převedení zdrojového kódu do kódu, který je čitelný počítačem. Gradle skripty pro budování jsou psány v programovacích jazycích Groovy a Kotlin DSL. Mezi nejznámější vývojová prostředí, která Gradle podporuje jsou Android Studio, Eclipse, IntelliJ IDEA, Visual Studio 2019 a XCode. [17] Používá se při tvorbě mobilních aplikací, až po tvorbu mikroskopických služeb a používají ho jak malé podniky, tak i velké firmy, například LinkedIn, Netflix, Adobe a Elastic NV. [18]

Mezi hlavní rysy Gradlu patří:

- **Flexibilita** – Gradle byl zvolen Googlem jako oficiální nástroj pro budování programu pro Android Studio (prostředí pro vývoj Androidových aplikací [19]), převážně díky tomu, že Gradle umožňuje velkou rozšiřitelnost a přizpůsobení podle vlastních potřeb. Příkladem je možnost použití kódu napsaného v jazycích C a C++, i když je výchozí jazyk pro vývoj v Android Studiu Java.
- **Výkonnost** – doba budování programu je jeden z nejvíce přímých způsobů, jak urychlit vývoj aplikace. Gradle optimalizuje výkonnost těmito způsoby:
    - **Inkrementalismus** – vyhýbá se zbytečné práci pomocí sledování vstupů a výstupů úkolů a spouští pouze to, co je nezbytné. Pokud možno, zpracovává pouze soubory, které se změnily.
    - **Mezipaměť budování** – pokud byla úloha už provedena na jiném počítači, Gradle může úplně přeskočit lokální spuštění a rovnou načíst výstupy úlohy z mezipaměti. Lokální mezipaměť budování lze použít pro úlohy provedeny v minulosti na stejném počítači.
    - **Gradle Démon** – dlouho žijící proces, který běží v pozadí a umožňuje znovupoužití předchozích budováních programu. [20]
- **Sken budování na webu** – Gradle poskytuje interaktivní uživatelské rozhraní, které je určeno k debuggování a optimalizaci budování.
- **Paralelní spuštění** – Gradle umožňuje paralelní spuštění úloh a úloh uvnitř, což má za následek rychlejší budování.
- **Časové limity úkolů** – každá úloha má svůj časový limit, který když přesáhne, bude terminována a umožní tak dokončení budování.
- **Nepřetržité budování** – pokud je Gradle úloha zapnutá v nepřetržitém módu, tak se automaticky hlídají změny vstupu pro tuto úlohu a kdykoliv se vstup změní, je úloha automaticky provedena.
- **Vyloučení úlohy** – jakákoliv úloha může být vyloučena ke spuštění. Když se toto vyloučení aplikuje, všechny úlohy, na kterých tato úloha závisí jsou automaticky vyloučeny taky, pokud nemají své vlastní závislosti.
- **Suchý běh –** možnost budování bez toho, aby se provedly instrukce uvnitř úloh. Slouží k tomu, aby se zjistilo, které úlohy budou spuštěny.
- **Pokračování budování i v případě chyby** – nepřestane budovat program, pokud narazí na chybu. Provádí všechny úkoly, které mají být provedeny, pokud byly všechny závislosti této úlohy dokončeny bez selhání. Díky tomu umožňuje objevit co nejvíce chyb v jediném budování se sumarizující chybovou zprávou na konci. [21]

2.5. Webový rámec JavaServer Faces

JSF je webový rámec založený na Javě, který slouží k usnadnění vývoje webových aplikací. Je obsažen v Jakarta platformě, což znamená, že je vestavěny do každého aplikačního serveru, který je v souladu s Jakartou. JSF slouží k vykonání dvou hlavních funkcí:

- **Generování uživatelského rozhraní** – typicky jako HTML, který je předán prohlížeči a následně zobrazen jako webová stránka. Toto uživatelské rozhraní je na straně serveru reprezentováno pomocí stromu komponentů. Elementy uživatelského rozhraní jsou namapovány do stromu komponentů. Uživatelské rozhraní, které vidí uživatel je generováno teprve tehdy, kdy je vykreslen tento strom komponentů. Oddělení stromu komponentů od uživatelského rozhraní umožňuje podporu více značkovacích jazyků a více prohlížečových prostředí. (například stolní počítač a mobil)
- **Reakce na události generované uživatelem** – na straně serveru vytváří posluchače, které čekají na vstup uživatele a následně na něj reagují. [22]

JSF stránka je typicky složena ze dvou části:

- **XHTML soubor** – značkovací soubor pro budování HTML stránek. Soubor se píše pomocí JSF štítků, například „<h:inputText />“ a pomocí výrazového jazyku k přístupu k tzv. Java Beanům, například „#{diagramView.model}“.
- **Java Bean** – Java třída značená anotací. Obsahuje veškerá data a metody používané v XHTML souboru a určuje, jak dlouho tyto data budou přístupná na serveru. [23]

```
@ManagedBean(name="metricView")
@ViewScoped
```

Obrázek 1: Příklad anotace Java Beanu

2.6. Sbírka komponentů PrimeFaces

PrimeFaces je jedna z nejpopulárnějších sbírek předem postavených komponentů, která rozšiřuje JSF rámec a umožňuje rychlý vývoj webových aplikací.

PrimeFaces je vývojáři upřednostňován hlavně díky těmto rysům:

- Obsahuje kolekci více než sto JSF komponentů pro zobrazení grafických rozhraní jako například kalendáře, obrázkové galérie, nebo HTML editor, který umožňuje uživateli přizpůsobení textu podle sebe.
- Předpřipravené kódy pro datové komponenty, které umožňuji vlastní implementaci třídění, filtrování a stránkování. Příklad takového komponentu je například tabulka, která se značí štítkem „<p:dataTable>“.
- Podpora exportu dat z tabulek do formátů XML, XLS, PDF a CSV.
- 35 předem definovaných motivů pro rozvržení stránky, včetně rámce pro vytváření vlastních motivů.
- Vylepšení a zjednodušení používání AJAXu (skupina sdružených technologií pro vytváření asynchronních webových aplikací [24]), oproti základnímu JSF.
- Knihovny pro vyvíjení mobilních aplikací a HTML5 grafických komponent. [23]

Je používán více než 5 milióny vývojářů ve Fortune 500 (seznam 500 nejvýdělečnějších společností v Spojených státech amerických [25]) společností, mezi které patří například Intel, eBay, UniCredit a mnoho dalších. [26]

```
<p:column headerText="Id" width="6%" style="text-align: right;">
    <p:cellEditor disabled="true">
        <f:facet name="output"><h:outputText value="#{metricTable.metricId}" /></f:facet>
        <f:facet name="input"><p:inputText id="idInput" value="#{metricTable.metricId}" st
    </p:cellEditor>
</p:column>
```

Obrázek 2: Ukázka PrimeFaces komponentu (definice sloupce v datové tabulce)

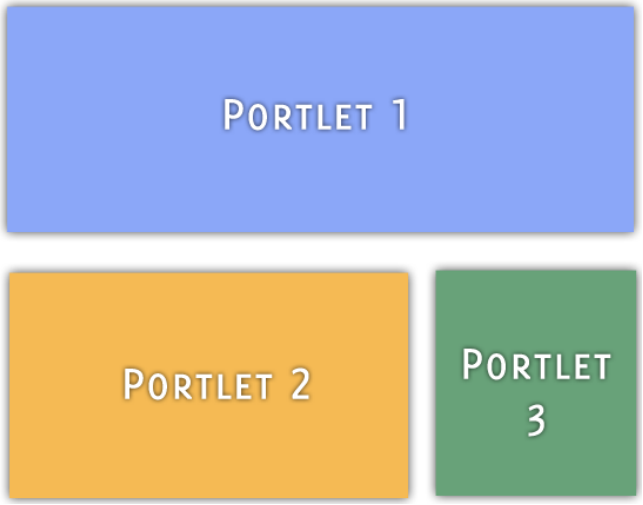
2.7. Platforma Liferay Portal

Je softwarová platforma s otevřeným kódem pro podnikové standardy založená na jazyce Java. Vývojářům poskytuje mnoho služeb pro usnadnění a zefektivnění práce, například:

- **Správa identity a řízení přístupu (bezpečnost aplikace)**:
  - Podporuje správu identity pomocí přihlášení přes Facebook, pokud uživatel chce, nemusí se nikde registrovat, stačí se přihlásit přes svůj Facebook účet.
  - Služby pro řešení správu uživatelů, autorizaci a autentizaci. Pomocí správy uživatelů lze vytvářet účty, skupiny, role, nebo politiky pro hesla. Zároveň lze data o uživatelích modifikovat.
  - Podpora vlastních definicí služeb, které mohou být spuštěny vzdáleně s větší mírou bezpečnosti.
- **Služby platformy:**
  - Sledování a správa uživatelské aktivity, přístup ke klíčovým statistikám aplikací a stránek, například počet kliků na stránce a nejdelší doba žádosti.
  - Podpora jednostránkových aplikací. Načítání pouze nezbytné části stránky při obnovení, což vede k rychlejšímu výkonu.
  - Podpora klasických Jakarta aplikací, vytváření a správa webů pro různé uživatele. Každý web obdrží vlastní stránky, systém pro správu obsahu a oprávnění. Admini mohou například měnit motivy, vkládat vlastní kódy a vytvářet menu.
  - Snadná správa a kontrola indexování a dalších funkcí vyhledávače.
- **Integrace a vzájemná spolupráce:**
  - Možnost importování Liferay Workspace do jakéhokoliv vývojového prostředí, díky kterému se usnadní vývoj Liferay modulů.
  - Vývojáři mezi sebou mohou prodávat, sdílet a stahovat témata, zásuvné moduly i celé aplikace pomocí Liferay Marketplace.
  - Konfigurovatelné a rozšiřitelné spojení do různých databázi, úložišti souborů a cloudovým službám. [27] [28]

2.7.1. Webová komponenta Portlet

Portlet je název pro webové aplikace uvnitř Liferay Portálu. Hlavní rozdíl mezi klasickou webovou aplikací a portlem je ten, že můžeme vytvořit více portletů jako samostatné celky a následně je poskládat. To znamená, že můžeme například vytvořit stránku z 3 různých portletů, kde jeden porlet bude kalendář pro události, druhý pro různé oznámení a třetí bude portlet se záložkami. Výhodou tohoto přístupu je, že jediná starost vývojáře je vývoj daných portletů a zbytek stránky, jako je třeba navigace, horní banner, popřípadě další jiné grafické rozraní jsou zpracovány jinými komponenty. Další výhodou je, že tyto portlety můžeme později přizpůsobovat podle sebe bez toho, abychom museli dělat úpravu kódu, například přemístění a změna velikosti, což by u klasických webových aplikací vyžadovalo manuální překódování. [28]



Obrázek 3: Příklad skládání portletů k vytvoření webové stránky [28]

2.7.2. Služba Service Builder

Důležitou součástí Liferay Portalu je Service Builder. Jedná se velmi robustní službu, pomocí které lze snadně vytvářet tabulky, modely a logiku uvnitř databáze. Konfigurace se provádí uvnitř souboru service.xml, který je automaticky vytvořen při založení projektu se Service Builderem. Obsahuje grafické rozhraní pro definice entit a rozhraní pro manuální psaní zdrojového kódu, který může vývojář použít k manuální tvorbě entit. [29]

```
<entity local-service="true" remote-service="false"
    name="Metric">
    <column name="metricId" primary="true" type="long"></column>
    <column name="name" type="String"></column>
    <column name="description" type="String"></column>
    <column name="properties" type="String"></column>
    <column name="severity" type="int"></column>
    <column name="satisfied" type="boolean"></column>
    <column name="data" type="String"></column>
    <column name="metricTypeId" type="long" />
    <column name="aspiceProcessId" type="long" />
    <finder return-type="Collection" name="MetricType">
        <finder-column name="metricTypeId" />
    </finder>
    <finder return-type="Collection" name="AspiceProcess">
        <finder-column name="aspiceProcessId" />
    </finder>
</entity>
```

Obrázek 4: Příklad definice entity uvnitř zdrojového kódu

K dnešnímu datu (18.03.2020) má Liferay přes 250 partnerů, mezi které patří například firmy Canon a Bosch. [30]

2.8. Nástroj pro správu databáze DBeaver

Je zdarma dostupný multiplatformní nástroj pro správu databáze s otevřeným kódem, určený pro vývojáře a administrátory databází. Pomocí DBeaveru lze provádět úpravu dat přes grafické rozhraní jako v klasických tabulkách a exportovat data do různých formátu. Podpora je specificky pro databáze, které mají JDBC ovladač (Java API pro přístup do jakéhokoliv druhu tabulkových dat, zejména data uložená v relační databázi [31]). Standardní verze DBeaveru poskytuje podporu pro více než 80 databází, mezi které patří MySQL, MariaDB a PostgreSQL. DBeaver je i dostupný jako zásuvný modul pro Eclipse IDE a lze ho přímo integrovat do vývojového prostředí, kde odpadá nutnost stahovat DBeaver jako samostatnou aplikaci. [32] [33]

	123 metricTypeId	ABC name	ABC type_	ABC description
1	6	Typ Metriky 1	Typ 1	Popis typu Metriky 1
2	7	Typ Metriky 2	Typ 2	Popis typu Metriky 2
3	8	Typ Metriky 3	Typ 3	Popis typu Metriky 3
4	9	Typ Metriky 4	Typ 4	Popis typu Metriky 4
5	10	Typ Metriky 5	Typ 5	Popis typu Metriky 5

Obrázek 5: Příklad zobrazení tabulky uvnitř DBeaveru

2.9. Databáze MariaDB

Jedná se o jednu z nejpopulárnějších relačních databází s otevřeným kódem. Je nástupnickou větví MySQL. Byla založena právě hlavními členy původního MySQL týmu, mezi které patří i sám zakladatel MySQL, momentálně je ale vyvíjená i komunitou.

Transakční koncept MariaDB je v souladu s ACID, díky kterému se zajišťuje, že data v databázi zůstávají přesná a konzistentní i přes jakékoli transakční selhání. [34] [35]

Ve srovnání s původním MySQL vyniká MariaDB v mnoha oblastech:

- Zlepšení rychlosti – mnohem rychlejší kontrola oprávnění u systémů s mnoha uživatelskými účty nebo mnoha databázemi.
- Rychlejší indexace – pro číselné indexy je hledání o 24% rychlejší, u znakových sloupců do 20 znaků je rychlejší o 60%.
- Rychlejší kontrola totožnosti dvou tabulek.
- Až 2x rychlejší a bezpečnější replikace databáze, včetně podpory paralelních replikací.
- Podpora více databázových systému pro ukládání dat. [36]

MariaDB je podporována i službami cloudových databázi, jako například Microsoft Azure, [37] Alibaba Cloud [38] a Amazon RDS. [39]

Mezi známé firmy využívající MariaDB patří Samsung, Nokia a Development Bank of Singapore. [40]

2.10. Webový server Apache Tomcat

Slouží jako webový server a kontejner Java Servletů (třída, která odpovídá na síťové požadavky, typicky HTTP požadavky, což umožňuje tvorbu dynamické webové stránky [41]) založený na jazyce Java s otevřeným kódem, který umožňuje spouštění webových aplikací. [42] Apache Tomcat standardně poskytuje HTTP konektor na internetový port 8080. Po uvedení serveru do chodu je pro napojení na tento port potřeba v jakémkoliv webovém prohlížeči vyhledat URL „http://localhost:8080". [43]

2.11. Verzovací systém Git

Je zdarma dostupný software, který slouží jako implementace systému správy verzí s otevřeným kódem. Byl navržen jak pro malé, tak i velké projekty, kde je potřeba kolaborace týmu a sloučení, záloha a sledování historie kódu.

Git realizuje správu verzí pomocí 3 základních funkcí:

- Commit – zachycení aktuálního stavu projektu a porovnání provedených změn. Každý commit má svůj kontrolní součet, pomocí kterého se vygeneruje unikátní ID commitu, díky kterému je zaručena integrita dat.
- Push – nahrání commitu do sdíleného adresáře. Povoleno jen tehdy, když má daný počítač nejaktuálnější verzi projektu.
- Pull – získání nejaktuálnější verze projektu a sloučení změn do lokálního adresáře.

Git operace jsou prováděny lokálně na počítači, což oproti centralizovaných systémů, které neustále musejí komunikovat se vzdáleným serverem poskytuje z hlediska rychlosti obrovskou výhodu. [44] [45]

3. PRAKTICKÁ ČÁST

V této části bakalářské práce popisuji průběh práce na projektu a rozvádím zde zadané úkoly a postup mých řešení.

3.1. Příprava

Jelikož jsem na odborné praxi používal vlastní zařízení, prvním nezbytným krokem před započetím práce byla příprava všech technologií, které jsem později potřeboval k plnění úloh zadaných vedoucím.

Jako první bylo zapotřebí stáhnutí a instalace Eclipse IDE pro podnikové účely s podporou Jakarty. Dále bylo potřeba stáhnout JDK 8, který je zapotřebí k umožnění vývoje a spuštění aplikací na bázi programovacího jazyka Java. Do Eclipse IDE bylo potřeba nainstalovat Liferay Portal, který měl i standardně obsažen Service Builder. Po implementaci Liferay Portalu bylo potřeba stáhnout a naimplementovat do Eclipse IDE webový server Apache Tomcat. Senior programátor mi poskytnul seznam všech závislostí, které bylo potřeba vložit do souboru „build.gradle“. Díky těmto závislostem se do Eclipse IDE přes budovací nástroj Gradle naimplementovalo a umožnilo používat Apache Log4j (nástroj pro vypisování logů ze serveru do konzole vývojového prostředí), PrimeFaces a JSF. Posledními kroky už jen bylo stažení a spuštění databáze MariaDB na počítači a stažení aplikace DBeaver pro možnost zobrazení této databáze.

3.2. Testovací projekt

Během prvních pár dnů ve firmě jsem měl za úkol vytvořit testovací projekt pro praktické obeznámení s technologiemi. Konkrétně se jednalo o tyto věci:

- Zprovoznění a spuštění webového serveru Tomcat.
- Vyzkoušení si tvorby uživatelského rozhraní pomocí xhtml včetně implementace různých komponentů PrimeFaces, jako například vytvoření a naplnění klasických a interaktivních grafů, různé zobrazení dat pomocí tabulek a diagramy.
- Aby se zajistila základní logika webové stránky a zobrazení nadefinovaných proměnných v xhtml souboru, bylo potřeba vytvořit Java třídu a definovat Java Bean.
- Dále pomocí Service Builderu obsaženého v Liferay vytvořit testovací tabulku a naplnit ji náhodnými daty.
- Nakonec se pomocí platformy Liferay Portal napojit na databázi a nakonfigurovat webovou stránku s použitím portletu, na kterým se vše zobrazilo a umožnilo otestování funkčnosti.

Jakmile vše bylo hotovo, senior programátor testovací projekt schválil a zadal první úkol ke konkrétnímu projektu.

3.3. Zadání projektu

Zadáním projektu bylo navrhnout systém „Metrics“ pro splnění standardu Automotive SPICE (model, který definuje procesy a osvědčené postupy pro vývoj softwaru a softwaru založeném zejména na automobilovém průmyslu, díky kterému lze organizovat práci na projektu [46]). Uvnitř tohoto systému bude možno vytvářet mnoho projektů, kde každý projekt bude mít jednu, nebo více metrik. Pojem metrika je zde myšlen jako souhrn požadavků zákazníka, zhodnocení kvality splnění těchto požadavků a posudek uspokojení zákazníka. Každá metrika se bude vázat na ASPICE Proces, což představuje vývoj software, který se hodnotí dle standardů ASPICE. Hlavním cílem tohoto systému bude umožnění sledování a zobrazení těchto metrik a jejich postupný vývoj v čase.

Zároveň zde bude správa uživatelů, kde každý uživatel bude představovat zákazníka, který se váže a má přístup k projektům, se kterými souvisí. Každý uživatel by měl mít možnost vytváření vlastních projektů a uvnitř projektů možnost vytváření vlastních metrik.

3.4. Návrh a implementace databáze

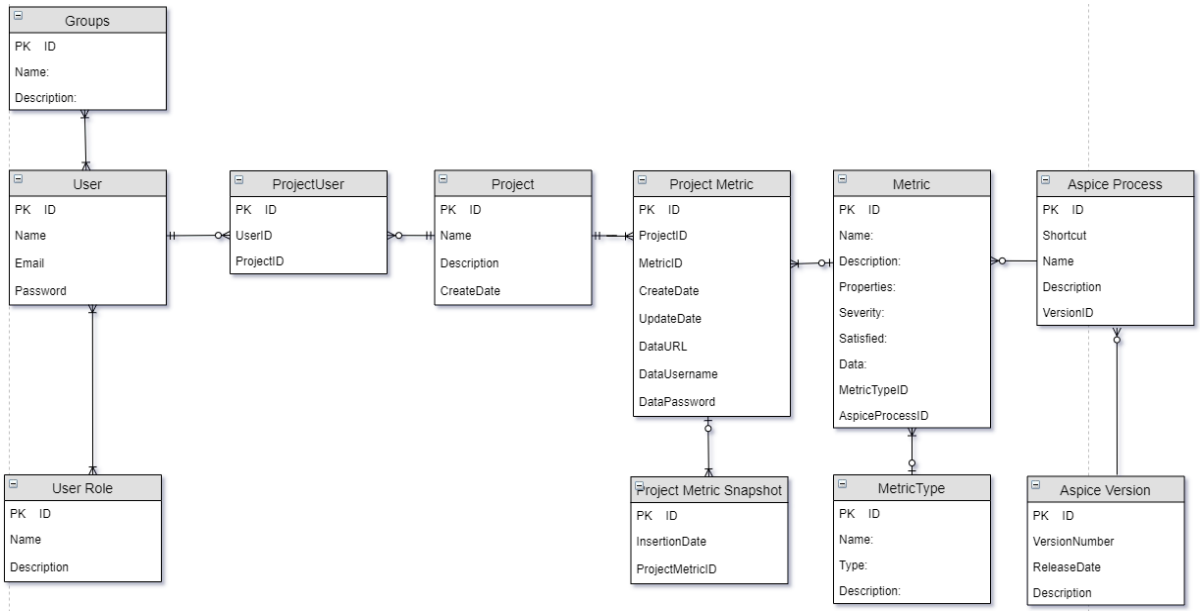
Nezbytným prvním a základním krokem pro projekt byl návrh a implementace databáze. Návrh databáze probíhal na stránce www.draw.io, která slouží jako editor pro vytváření vývojových diagramů.

3.4.1. Návrh databáze

Databáze obsahuje celkem 11 různých tabulek (zobrazeny na obrázku 6), mezi kterými jsem určil relační vztahy na základě toho, jak budou využívány:

- Mezi tabulkami „User“ (Uživatel) a „Groups“ (Skupiny) je vztah Více k Více, což znamená, že jeden uživatel může mít více skupin a jedna skupina může být součástí více uživatelů.
- Tabulka „User Role“ (Role Uživatele) má taktéž mezi tabulkou „User“ také vztah Více k Více jelikož jeden uživatel může nabýt více rolí a jednu roli může sdílet více uživatelů, mohou ale existovat i samostatně bez napojení.
- Mezi tabulkou „User“ a „Project User“ (Uživatel Projektu) byl zvolen vztah Jeden Povinný a Mnoho Volitelných. Znamená to, že aby existoval uživatel projektu, musí první existovat uživatel. Pokud je tato podmínka splněna, uživatel může být zařazen mezi více různých projektů.
- Tabulky Project User a Project (Projekt) má taktéž mezi sebou vztah Jeden Povinný a Mnoho Volitelných. Aby existoval uživatel projektu, musí existovat nějaký projekt. Pokud existuje projekt, může, ale nemusí mít mnoho různých uživatelů tohoto projektu.
- Tabulky Project a Project Metric (Projektová Metrika) mezi sebou mají vztah Jeden Povinný k Více Povinným. Znamená to, že když existuje projekt, tak tím pádem musí mít i své projektové metriky.
- Tabulky Project Metric a Project Metric Snapshot (Snímek Projektové Metriky) mezi sebou mají vztah Jeden Povinný k Více Povinným. Pokud tedy existuje Projektová Metrika, musí mít své Snímky.
- Mezi tabulkami Project Metric a Metric (Metrika) byl určen vztah Více Povinných k Jednomu Volitelnému, protože Projektová Metrika může existovat i bez Metriky. Pokud ale existuje Metrika, musí být napojená na alespoň jednu Projektovou Metriku.

- Tabulky Metric a Metric Type (Typ Metriky) mají mezi sebou vztah Více Povinných k Jednomu Volitelnému. Metrika nemusí, ale může mít svůj Typ. Pokud ale existuje typ, musí být součástí alespoň jedné Metriky.
- Tabulky Metric a ASPICE Process (Proces ASPICE) mají mezi sebou vztah Nula ku Více Volitelným. Oba mohou existovat samostatně, ale Proces ASPICE může, ale nemusí, být součástí Více Metrik.
- Poslední vztah je mezi tabulkami ASPICE Process a ASPICE Version (verze ASPICE), opět byl zvolen vztah Nula ku Více Volitelným. Oba mohou existovat samostatně, ale verze ASPICE může, ale nemusí, být součástí více ASPICE Procesů.



Obrázek 6: Návrh tabulek a vztahů v databázi

3.4.2. Implementace databáze

Implementace databáze byla realizována pomocí služby Service Builder a MariaDB. Na začátek bylo potřeba nadefinovat a vytvořit tabulky Metric, Metric Type, ASPICE Process a ASPICE Version.

Základem bylo založit novou databázi a vytvořit napojení na tuto databázi. Toho se dosáhlo pomocí aplikace DBeaver, která pro tyhle účely poskytuje jednoduché uživatelské rozhraní.

Dalším krokem bylo vytvoření tabulek a vztahů mezi nimi uvnitř souboru „service.xml“. (obrázek 7)

Posledním krokem bylo použití funkce „build-service“ obsaženou v Service Builderu, která z definic uvnitř souboru service.xml vytvořila veškeré modely a základní implementační třídy, včetně persistencí a jejich implementace.

```
<namespace>metrics</namespace>
<entity local-service="true" remote-service="false"
    name="Metric">
    <column name="metricId" primary="true" type="long"></column>
    <column name="name" type="String"></column>
    <column name="description" type="String"></column>
    <column name="properties" type="String"></column>
    <column name="severity" type="int"></column>
    <column name="satisfied" type="boolean"></column>
    <column name="data" type="String"></column>
    <column name="metricTypeId" type="long" />
    <column name="aspiceProcessId" type="long" />
    <finder return-type="Collection" name="MetricType">
        <finder-column name="metricTypeId" />
    </finder>
    <finder return-type="Collection" name="AspiceProcess">
        <finder-column name="aspiceProcessId" />
    </finder>
</entity>
<entity name="MetricType" local-service="true"
    remote-service="false">
    <column name="metricTypeId" type="long" primary="true"></column>
    <column name="name" type="String"></column>
    <column name="type" type="String"></column>
    <column name="description" type="String"></column>
</entity>
<entity name="AspiceVersion" local-service="true"
    remote-service="false">
    <column name="aspiceVersionId" type="long" primary="true"></column>
    <column name="versionNumber" type="int"></column>
    <column name="releaseDate" type="Date"></column>
    <column name="description" type="String"></column>
</entity>
<entity name="AspiceProcess" local-service="true"
    remote-service="false">
    <column name="aspiceProcessId" type="long" primary="true"></column>
    <column name="name" type="String"></column>
    <column name="description" type="String"></column>
    <column name="shortcut" type="String"></column>
    <column name="aspiceVersionId" type="long" />
    <finder return-type="Collection" name="AspiceVersion">
        <finder-column name="aspiceVersionId" />
    </finder>
</entity>
```

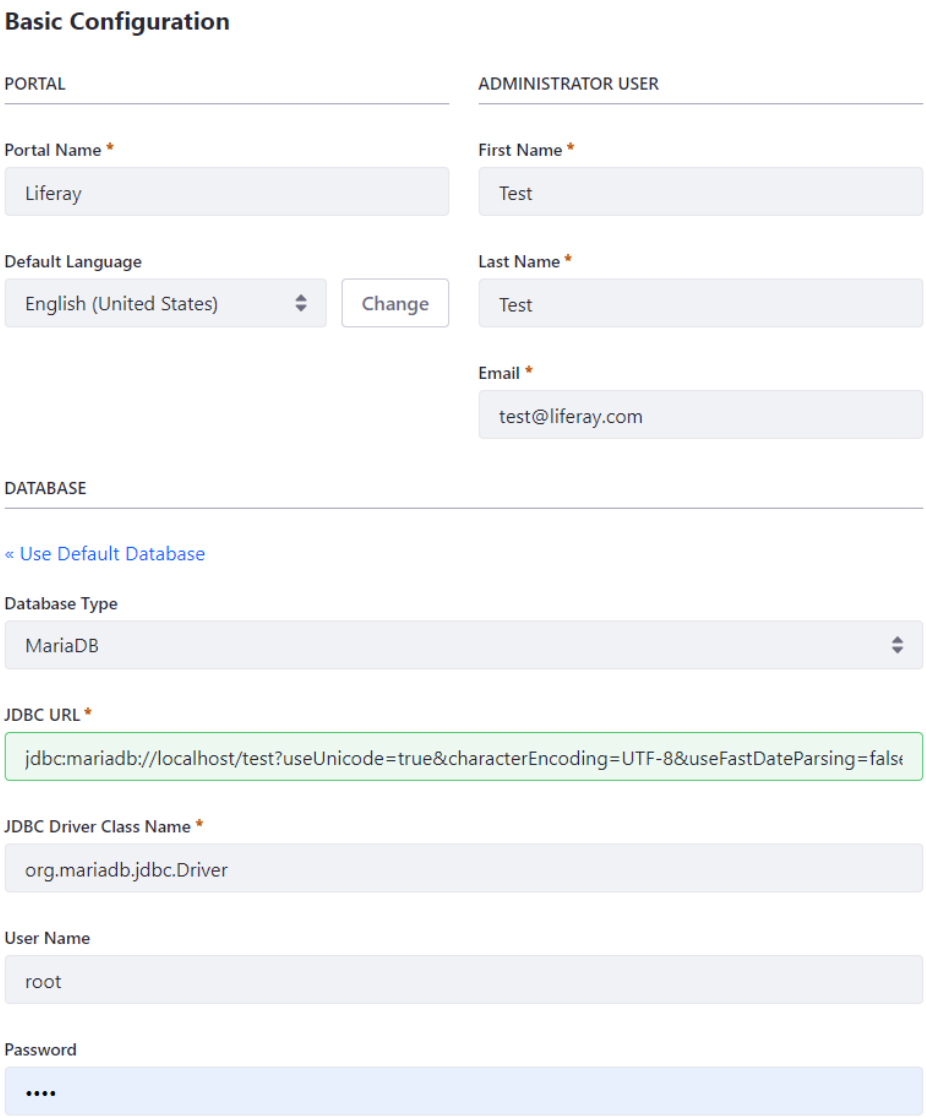
Obrázek 7: Implementace tabulek a vztahů mezi nimi pomocí Service Builderu (zdrojový kód)

3.5. Základní funkčnost webové stránky

Pro implementaci projektu bylo potřeba zprovoznit Liferay Portal, který běží na webovém serveru Tomcat. Pro napojení na tento server bylo potřeba se přes webový prohlížeč napojit na stránku „http://localhost:8080”, neboli na port 8080 daného počítače, na kterém běžel server a uvnitř Liferay Portalu nadefinovat propojení s databází. Na stránce se zatím nezobrazovalo nic, ale fungovala.

3.5.1. Konfigurace Liferay

Při prvním startu serveru je vždy potřeba nakonfigurovat napojení k databázi, jméno Liferay Portálu a uživatele, který bude v roli admina portálu. Vyplnil jsem informace o databázi a jelikož databáze při vývoji běží pouze lokálně, stačilo ponechat základní údaje pro nastavení admina.



Obrázek 8: Konfigurace Liferay Portalu

3.5.2. Projekt z pohledu Eclipse

Projekt uvnitř Eclipse byl reprezentován modulem (složkou) s názvem „metrics“. Uvnitř tohoto modulu sídlily dvě hlavní složky – jedna s názvem „views“, uvnitř které byly xhtml soubory, kde se definovaly jednotlivé stránky a další pro Java třídy pro implementaci kódu.

Zároveň byly s každým sestavením Service Builderu tvořeny, popřípadě aktualizovány moduly „metrics.servicebuilder-api“ a „metrics.servicebuilder-service“, uvnitř kterých se automaticky vytvářely Java třídy, které se využívaly pro komunikaci s databází.

3.6. Rozhraní a kód tabulky Metric

3.6.1. Základní funkčnost

Po implementaci databáze bylo potřeba vytvořit uživatelské rozhraní a logiku pro tabulku Metrik. Pro implementaci uživatelského rozhraní jsem založil nový xhtml soubor s názvem „metricView.xhtml“ a novou Java třídu, z které se pro tohle rozhraní získával Java Bean, který jsem pojmenoval „metricView“. Uvnitř nové třídy bylo potřeba nadefinovat veškeré proměnné, které bude potřeba využít v xhtml souboru pro implementaci základní, či pokročilé logiky. (obrázek 9) Ve vztahu mezi Java Beanem a xhtml platí pravidlo, že proměnné je zapotřebí zapouzdřit jako „private“ a pro získání jejich hodnoty, či nastavení jejich hodnoty poskytnout „getter“ a „setter“ metody, jinak tyto proměnné budou vyhodnoceny jako nepřístupné (i kdyby byly zapouzdřené jako „public“) a aplikace nebude fungovat.

```
@ManagedBean(name="metricView")
@ViewScoped
public class MetricView implements Serializable{
    private List<Metric> selectedMetrics;
    private List<Metric> metrics;
    private Metric selectedMetric;
    private Metric metric;
    private String name, description, properties, data;
    private int severity;
    private boolean satisfied;

    private long metricTypeId;
    private long aspiceProcessId;
    private String addDialogHeader;
    private String addDialogBottomCommandButton;

    private String metricTypeName;
    private String aspiceProcessName;
```

Obrázek 9: Část kódu Java třídy pro Metric tabulku s Java Beanem „metricView“

Dalším krokem bylo v „metricView.xhtml“ vytvoření rozhraní pro zobrazení tabulky. Pro zobrazení tabulky jsem využil PrimeFaces komponentu „p:dataTable“, kde jsem nadefinoval všechny sloupce tabulky a základní logiku pro označení řádků (obrázek 10).

```
<p:dataTable id="metrics" var="metricTable" value="#{metricView.metrics}" editable="true"
        selectionMode="multiple" selection="#{metricView.selectedMetrics}" rowKey="#{metricTable.metricId}">
    <f:facet name="header">
        Metrics

        <p:commandButton id="toggler" type="button" value="Columns"  style="float:right" icon="pi pi-align-justify" />
        <p:columnToggler datasource="metrics" trigger="toggler" />
    </f:facet>
    <p:ajax event="rowEdit" listener="#{metricView.onRowEdit}" update=":metricForm:metricMsgs" />
    <p:ajax event="rowEditCancel" listener="#{metricView.onRowCancel}" update=":metricForm:metricMsgs" />

    <p:column headerText="Id" width="6%" style="text-align: right;">
        <p:cellEditor disabled="true">
            <f:facet name="output"><h:outputText value="#{metricTable.metricId}" /></f:facet>
            <f:facet name="input"><p:inputText id="idInput" value="#{metricTable.metricId}" style="width:100%"/></f:facet>
        </p:cellEditor>
    </p:column>

    <p:column headerText="Name">
        <p:cellEditor>
            <f:facet name="output"><h:outputText value="#{metricTable.name}" /></f:facet>
            <f:facet name="input"><p:inputText id="nameInput" value="#{metricTable.name}" style="width:100%"/></f:facet>
        </p:cellEditor>
    </p:column>
    <p:column headerText="Description">
        <p:cellEditor>
            <f:facet name="output"><h:outputText escape="false" value="#{metricTable.description}" /></f:facet>
            <f:facet name="input"><p:inputText id="descriptionInput" value="#{metricTable.description}" style="width:100%"
        </p:cellEditor>
    </p:column>
    <p:column headerText="Properties">
        <p:cellEditor>
            <f:facet name="output"><h:outputText value="#{metricTable.properties}" /></f:facet>
            <f:facet name="input"><p:inputText id="propertiesInput" value="#{metricTable.properties}" style="width:100%"/>
        </p:cellEditor>
    </p:column>
    <p:column headerText="Severity" width="7%" style="text-align: center;">
        <p:cellEditor>
            <f:facet name="output"><h:outputText value="#{metricTable.severity}" /></f:facet>
            <f:facet name="input"><p:inputText id="severityInput" value="#{metricTable.severity}" style="width:100%"/></f:
        </p:cellEditor>
    </p:column>
    <p:column headerText="Satisfied" width="9%" style="text-align: center;">
        <p:cellEditor>
            <f:facet name="output"><h:outputText value="#{metricTable.satisfied}" /></f:facet>
            <f:facet name="input"><p:selectBooleanButton value="#{metricTable.satisfied}" onLabel="true" offLabel="false"
        </p:cellEditor>
    </p:column>
    <p:column headerText="Data">
        <p:cellEditor>
            <f:facet name="output"><h:outputText value="#{metricTable.data}" /></f:facet>
            <f:facet name="input"><p:inputText id="dataInput" value="#{metricTable.data}" style="width:100%"/></f:facet>
        </p:cellEditor>
```

Obrázek 10: Část kódu definice tabulky uvnitř xhtml souboru

Zatím chyběla možnost pro interaktivní přidání, úpravu a mazání nových řádků. Testovací data jsem zatím přidával přes DBeaver a vše se úspěšně zobrazovalo na stránce. (obrázek 11)

Metrics

Columns

Id	Name	Description	Properties	Severity	Satisfied	Data	Metric Type	Aspice Process
7	Metrika 1	Popis Metriky 1	Vlastnosti Metriky 1	2	false	Data Metriky 1	asdf	ereret
8	Metrika 2	Popis Metriky 2	Vlastnosti Metriky 2	3	true	Data Metriky 2	yxcv	vdsvxcv
9	Metrika 3	Popis Metriky 3	Vlastnosti Metriky 3	1	true	Data Metriky 3	yxcv	vbcnvbn
10	Metrika 4	Popis Metriky 4	Vlastnosti Metriky 4	3	true	Data Metriky 4	ert	vbcnvbn
11	Metrika 5	Popis Metriky 5	Vlastnosti Metriky 5	3	false	Data Metriky 5	rthrth	hztut
12	Metrika 6	Popis Metriky 6	Vlasnosti Metriky 6	1	true	Data Metriky 6	ert	ereret
13	Metrika 7	Popis Metriky 7	Vlastnosti Metriky 7	0	false	Data Metriky 7	yxcv	hztut

Obrázek 11: Základní zobrazení tabulky s testovacími daty na stránce

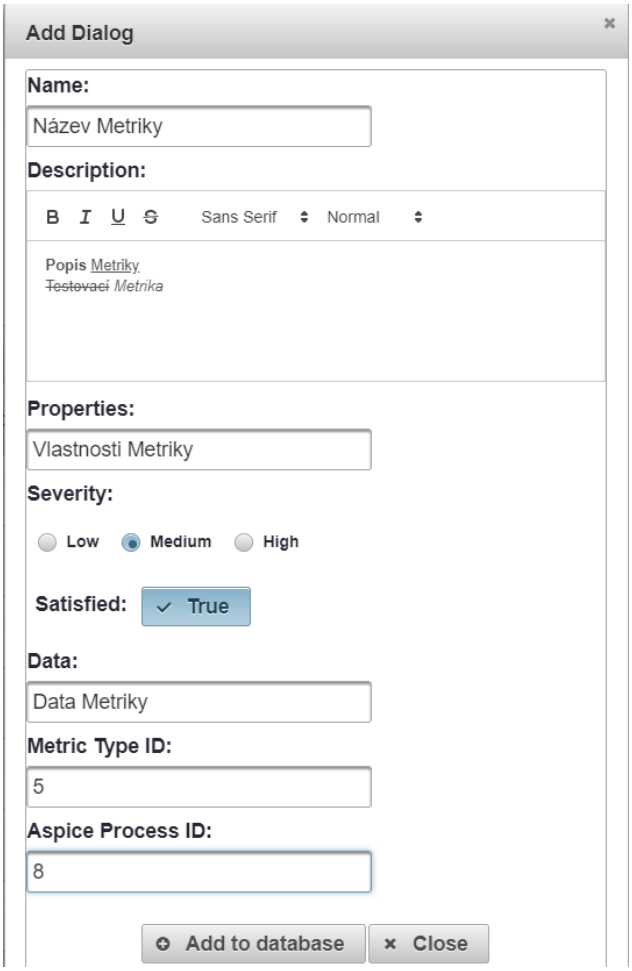
3.7. Vytváření, úprava a mazání metrik

Celá aplikace by se neobešla bez toho, kdyby pro našeho klienta nebyla možnost do jakékoliv tabulky přidat, upravit či mazat řádky.

3.7.1. Vytváření metrik

Pro vytváření nových metrik jsem pod tabulkou přidal tlačítko s nápisem „Add Metric“ (přidat metriku). Funkce tohoto tlačítka slouží k otevření dialogu, který umožní uživateli přidat a nadefinovat novou metriku.

Tento dialog (obrázek 12) obsahuje všechny sloupce, které v tabulce metrik existují, tzn. samostatná editovatelná políčka pro jméno, popis, vlastnosti, důležitost, data, zda byla metrika splněna, její Typ a její ASPICE proces.



Obrázek 12: Dialog pro přidaní metrik

Bylo však potřeba zajistit správný formát těchto vstupů. Zatímco jméno, vlastnosti a data metriky jsou jednoduchý řetězec znaků, tak samo je i popis, ten by ale měl být pro uživatele více přizpůsobitelný. V praxi to znamená, že uživatel by měl mít možnost v popisu provádět stejné úpravy vět, jako například v e-mailu, to znamená možnost podtržení, kurzívu, zvýraznění, velikost a typ písma. Díky PrimeFaces jsem měl možnost tohle specifické přizpůsobení jednoduše implementovat. Dále bylo potřeba, aby důležitost byla omezena pouze na tři vstupy – malá, střední a velká. To znamená, že jsem musel přidat tři tzv. rádiová tlačítka s jednotlivými názvy, kde ale každý měl svou číselnou hodnotu, jelikož je v tabulce důležitost označena číselným typem. Dalším vstupem je jednoduché booleovské políčko, které určuje, zda je metrika splněna a jediné povolené vstupy jsou pravda a nepravda. Zbývají už jen dvě políčka – Typ Metriky a ASPICE Proces. Jelikož se prozatím neřešilo propojení těchto tabulek, vstup pro tato políčka byl zatím jednoduchý řetězec. V pozdější kapitole se věnuji řešení problematiky propojení těchto tabulek a správného vstupu v dialogu.

```
<p:dialog id="addmetric" header="#{metricView.addDialogHeader}" widgetVar="addMetricsDialog" modal="true"
showEffect="fade" hideEffect="fade" resizable="true" width="auto" height="auto" closeOnEscape="true">
    <p:ajax event="close" update="addmetric"/>
    <p:scrollPanel style="height:1200px;width:500px;" mode="native">
        <p:outputPanel id="addMetricsDetail" style="text-align:left;">
            <p:outputLabel for="name" value="Name: "/><br/>
            <p:inputTextarea id="name" rows="1" cols="30" value="#{metricView.name}"/><br/>
            <p:outputLabel for="description" value="Description: " /><br/>
            <p:textEditor id="description" widgetVar="editor2" value="#{metricView.description}" height="120" style=
                <f:facet name="toolbar">
                     <span class="ql-formats">
                        <button class="ql-bold"></button>
                        <button class="ql-italic"></button>
                        <button class="ql-underline"></button>
                        <button class="ql-strike"></button>
                    </span>
                    <span class="ql-formats">
                        <select class="ql-font"></select>
                        <select class="ql-size"></select>
                    </span>
                </f:facet>
            </p:textEditor>
            <p:outputLabel for="properties" value="Properties: "/><br/>
            <p:inputTextarea id="properties" rows="1" cols="30" value="#{metricView.properties}"/><br/>
            <p:outputLabel for="severity" value="Severity: " />
            <h:panelGrid columns="2" style="margin-bottom:10px" cellpadding="5">
                <p:selectOneRadio id="severity" value="#{metricView.severity}" unselectable="true">
                    <f:selectItem itemLabel="Low" itemValue="1" />
                    <f:selectItem itemLabel="Medium" itemValue="2" />
                    <f:selectItem itemLabel="High" itemValue="3" />
                </p:selectOneRadio>
            </h:panelGrid>
            <h:panelGrid columns="2" style="margin-bottom:10px" cellpadding="5">
            <p:outputLabel value="Satisfied: " />
            <p:selectBooleanButton id="satisfiedVal" value="#{metricView.satisfied}" onLabel="True" offLabel="False"
            onIcon="pi pi-check" offIcon="pi pi-times" style="width:auto"/>
            </h:panelGrid>
            <p:outputLabel for="data" value="Data: "/><br/>
            <p:inputTextarea id="data" rows="1" cols="30" value="#{metricView.data}"/><br/>
            <p:outputLabel for="metricType" value="Metric Type ID: "/><br/>
            <p:inputTextarea id="metricType" rows="1" cols="30" value="#{metricView.metricTypeId}"/><br/>
            <p:outputLabel for="aspiceProcess" value="Aspice Process ID: "/><br/>
            <p:inputTextarea id="aspiceProcess" rows="1" cols="30" value="#{metricView.aspiceProcessId}"/><br/>
            <div style="text-align:center;padding-top:20px;">
```

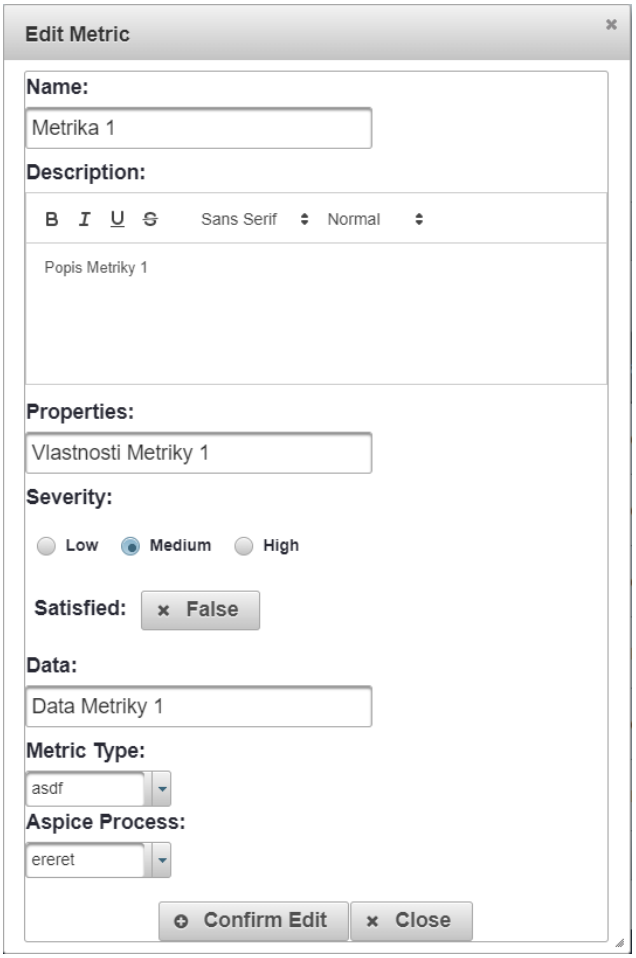
Obrázek 13: Část zdrojového kódu pro dialog

3.7.2. Úprava metrik

Dalším úkolem bylo zajistit, aby měl uživatel možnost jednoduše upravit jakoukoliv metriku v tabulce. Pro tuto funkci jsem do každého řádku přidal sloupeček, který obsahuje ikonku lupy. (obrázek 14). Po kliknutí na lupu jsem využil znovupoužití kódu, kde se otevře stejný dialog, jako dialog pro vytvoření nové metriky. Dialogu však nastavím jiný nadpis, a tlačítko „Add to database“ (přidat do databáze) změním na „Confirm edit“ (potvrdit úpravu), které po kliknutí zavolá funkci pro úpravu metriky. Všechna políčka se automaticky předvyplní z řádku, na který uživatel kliknul. (obrázek 15) Zároveň bylo potřeba naprogramovat logiku pro úpravu dané metriky uvnitř tabulky po potvrzení uživatelem.

Id	Name	Description	Properties	Severity	Satisfied	Data	Metric Type	Aspice Process	
7	Metrika 1	Popis Metriky 1	Vlastnosti Metriky 1	2	false	Data Metriky 1	asdf	ereret	

Obrázek 14: Řádek s lupou



Obrázek 15: Dialog po kliknutí na lupu (úpravu)

3.7.3. Mazání metrik

Další podstatnou funkcí je mazání metrik. Mazání metrik jsem vyřešil povolením tzv. „multiselectu“ (možnost vybrání více řádku v tabulce), kde uživatel označí řádky, které bude chtít vymazat, tyto řádky se uloží do proměnné typu list v Java Beanu pro metriky a jakmile uživatel klikne na tlačítko Smazat, zavolá se funkce, do které jsem nakódoval logiku pro smazání vybraných metrik a aktualizaci tabulky.

3.7.4. Export dat

Pokud uživatel potřebuje exportovat tabulku do známých formátů, jako je PDF a CSV, přidal jsem možnost pro realizaci tohoto exportu. (obrázek 16 a 17)

Lze i export přizpůsobit podle sebe, pokud potřebuje uživatel exportovat jen část tabulky, stačí označit tu část tabulky, kterou potřebuje exportovat a do PDF, či CSV se zapíšou jen zvolená data.

Id	Name	Type	Description
6	Typ Metriky 1	Typ 1	Popis typu Metriky 1
7	Typ Metriky 2	Typ 2	Popis typu Metriky 2
8	Typ Metriky 3	Typ 3	Popis typu Metriky 3
9	Typ Metriky 4	Typ 4	Popis typu Metriky 4
10	Typ Metriky 5	Typ 5	Popis typu Metriky 5
11	Typ Metriky 6	Typ 6	Popis typu Metriky 6
12	Typ Metriky 7	Typ 7	Popis typu Metriky 7

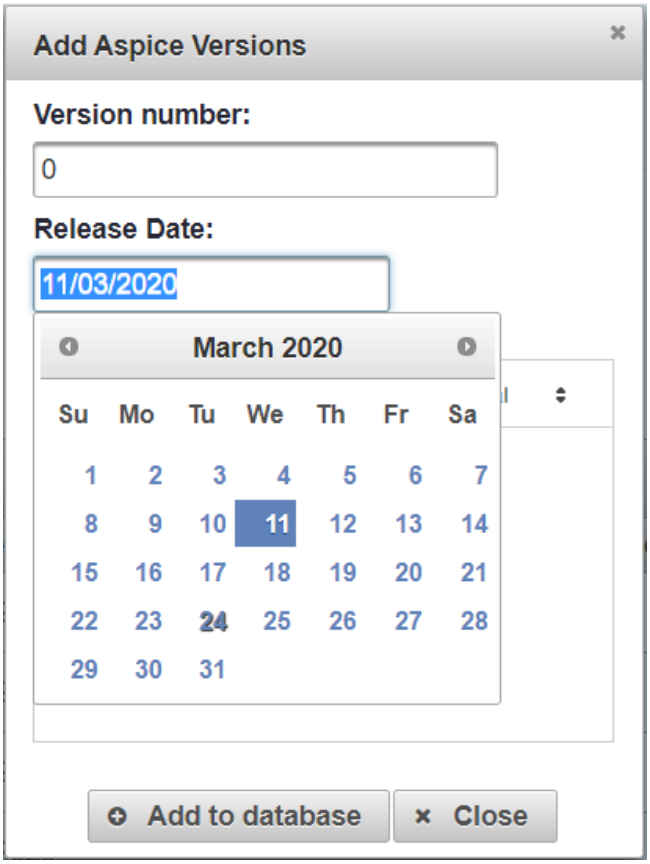
Obrázek 16: Export tabulky do formátu PDF

	A	B	C	D	E	F
1	Id	Name	Type	Description		
2	6	Typ Metriky 1	Typ 1	Popis typu Metriky 1		
3	7	Typ Metriky 2	Typ 2	Popis typu Metriky 2		
4	8	Typ Metriky 3	Typ 3	Popis typu Metriky 3		
5	9	Typ Metriky 4	Typ 4	Popis typu Metriky 4		
6	10	Typ Metriky 5	Typ 5	Popis typu Metriky 5		
7	11	Typ Metriky 6	Typ 6	Popis typu Metriky 6		
8	12	Typ Metriky 7	Typ 7	Popis typu Metriky 7		

Obrázek 17: Export tabulky do formátu CSV

3.7.5. Stránky Metric Type, ASPICE Process a ASPICE Version

Tak samo, jak jsem vytvářel rozhraní a logiku pro stránku Metrics, jsem vytvářel i pro stránky Metric Type, ASPICE Process a ASPICE Version. U stránky ASPICE Version bylo oproti ostatním stránkám nutné přidat novou PrimeFaces komponentu „<p:calendar/>“ (obrázek 18) do pole s Release Date (datum vydání), kde jsem musel i nastavit správný formát zápisu, který musel odpovídat výchozím konvencím MariaDB databáze pro zápis datumů.



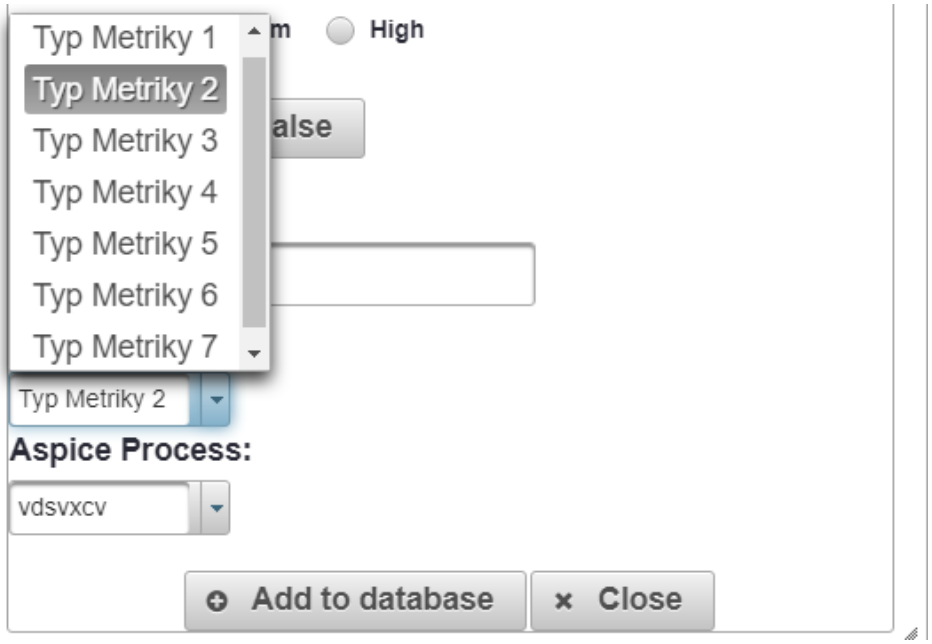
Obrázek 18: ASPICE Version dialog

3.8. Propojení tabulek

Další podstatnou části projektu bylo zajistit propojení tabulek, specificky při přidávání, mazání a úpravě záznamů v databázi.

3.8.1. Propojení při vytváření

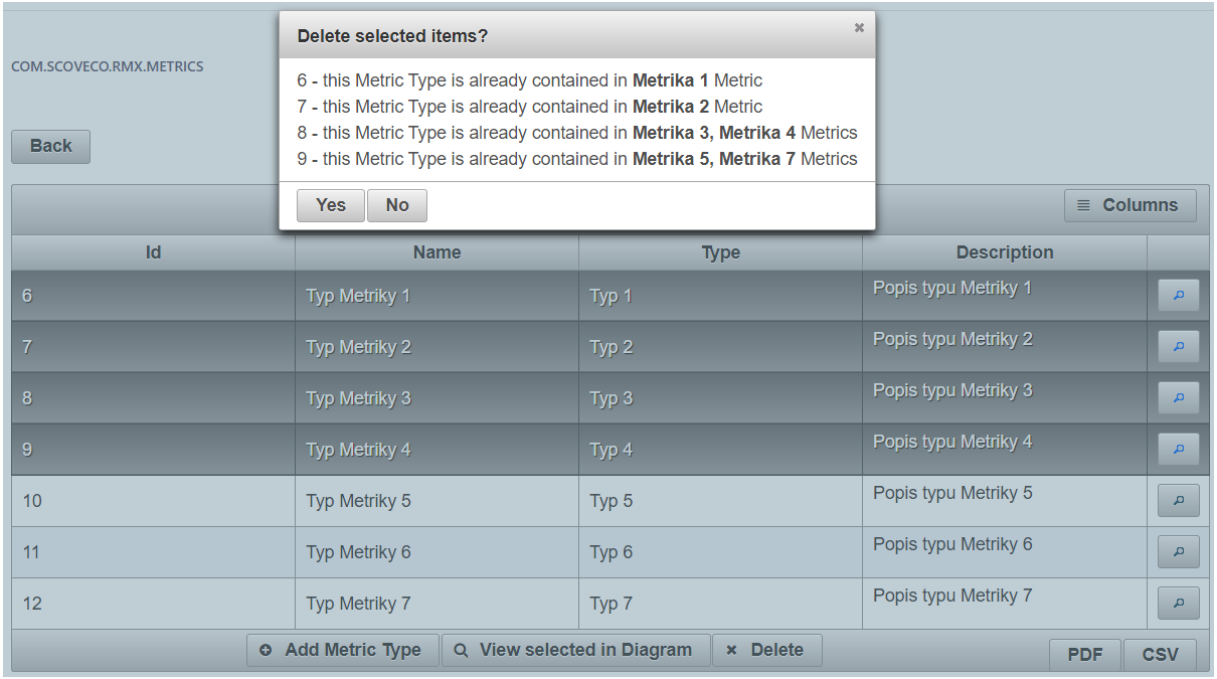
Při vytváření nových řádků pro tabulku metrik bylo potřeba pro uživatele u sloupců, které značí propojení mezi jinou tabulkou zajistit jasný vstup, který odpovídá reálnému stavu databáze. Implementaci jsem zajistil tak, že vždy získám všechny řádky z dané tabulky a z těchto řádků vyberu všechna jména, která nahraji do listu, který následně zobrazuji jako rozbalovací nabídku, z které si uživatel zvolí jednu položku. To samé jsem udělal i pro úpravu řádku, kde si uživatel z rozbalovací nabídky zvolí jinou položku a aktualizuje řádek. Toto jsem provedl pro všechny tabulky, na které se vztahují propojení.



Obrázek 19: Rozbalovací nabídka při vytváření nového řádku

3.8.2. Propojení při mazání

Pokud chce uživatel smazat řádek, který je propojen s jinou tabulkou, vyskočí mu upozornění, že dojde k narušení tohoto propojení. Docílil jsem toho vytvořením nové metody, do které posílám každý vybraný prvek a na základě toho se vrací buď prázdný řetězec, nebo řetězec s názvy objektů, uvnitř kterých je řádek obsažen.



Obrázek 20: Upozornění při mazání propojených řádků

```
public String checkIfAspiceProcessExistsInMetrics(AspiceProcess aspiceProcess) {
    List <Metric> metricsWhichContainThisAspiceProcess = MetricLocalServiceUtil.getAllMetrics()
            .stream().filter(metric -> metric.getAspiceProcessId() == aspiceProcess.getAspiceProcessId()).collect(Collectors.toList());

    String message = "";
    if (!metricsWhichContainThisAspiceProcess.isEmpty()) {
        message = " - this Aspice Process is already contained in <strong>" + metricsWhichContainThisAspiceProcess.get(0).getName();
        if (metricsWhichContainThisAspiceProcess.size() > 1) {
            for (int i = 1; i < metricsWhichContainThisAspiceProcess.size(); i++) {
                message += ", " + metricsWhichContainThisAspiceProcess.get(i).getName();
            }
        }
        message += "</strong> Metric" + (metricsWhichContainThisAspiceProcess.size() > 1 ? "s" : "");
    }

    return message;
}
```

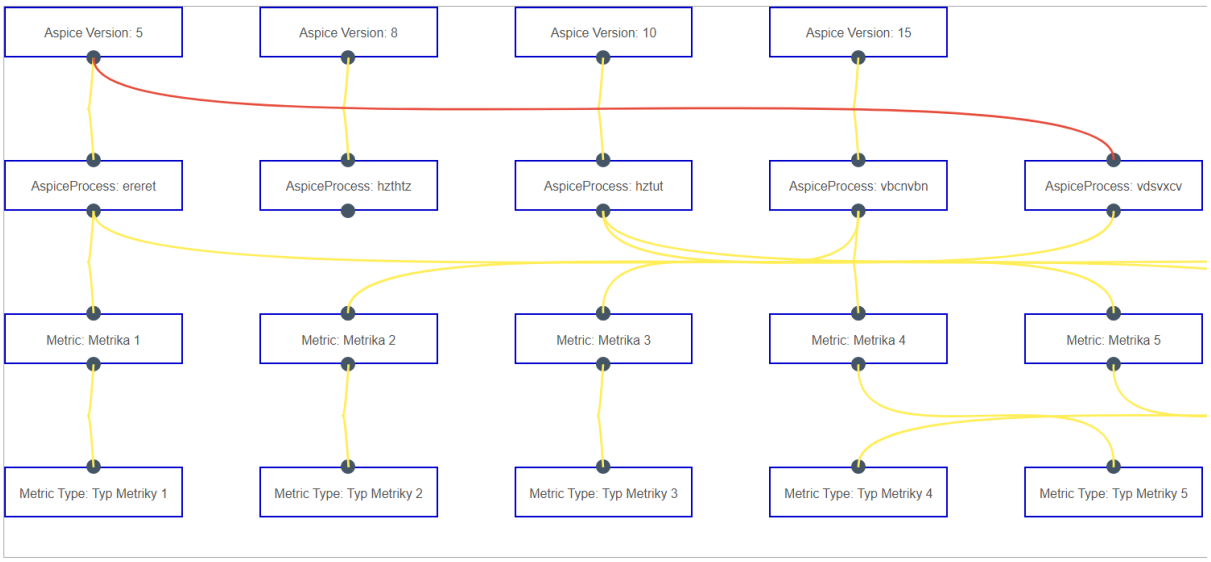
Obrázek 21: Funkce pro kontrolu výskytu v jiné tabulce

3.9. Zobrazení databáze a vztahů mezi tabulkami pomocí diagramu

Dalším úkolem bylo vytvořit grafické zobrazení dané databáze včetně propojení tabulek a jednotlivých záznamů uvnitř.

3.9.1. Implementace diagramu

Pro zobrazení diagramu jsem nejprve vytvořil novou xhtml stránku, do které jsem přidal PrimeFaces komponentu „Diagram“. Dále jsem udělal novou třídu, kde jsem naprogramoval logiku zobrazení jednotlivých entit a propojení mezi nimi. Pro tento diagram jsem i nadefinoval vlastní CSS (slouží pro vlastní přizpůsobení zobrazení různých komponent) styl, konkrétně tloušťku a barvu ohraničení, zarovnání textu, barva propojení a změna barvy na červenou při najetí myší na určité propojení. (tzv. „hover effect“)



Obrázek 22: Diagram pro zobrazení databáze a propojení tabulek

```
List<Metric> allMetrics = MetricLocalServiceUtil.getAllMetrics();
List<MetricType> allMetricTypes = MetricTypeLocalServiceUtil.getAllMetricTypes();
List<AspiceProcess> allAspiceProcesses = AspiceProcessLocalServiceUtil.getAllAspiceProcesses();
List<AspiceVersion> allAspiceVersions = AspiceVersionLocalServiceUtil.getAllAspiceVersions();

List<Element> metricsElements = new ArrayList<Element>();
List<Element> metricTypesElements = new ArrayList<Element>();
List<Element> aspiceProcessesElements = new ArrayList<Element>();
List<Element> aspiceVersionsElements = new ArrayList<Element>();

List<Map<Object, Element>> relationList = new ArrayList<Map<Object, Element>>();

for (int i = 0; i < allMetrics.size(); i++) {
    Metric currMetric = allMetrics.get(i);
    metricsElements.add(new Element("Metric: " + currMetric.getName(), String.valueOf(20*i)+"em", "24em"));

    metricsElements.get(metricsElements.size()-1).addEndPoint(new DotEndPoint(EndPointAnchor.BOTTOM));
    if (allMetricTypes.stream().anyMatch(metricType -> metricType.getMetricTypeId() == currMetric.getMetricTypeId()))
        metricsElements.get(metricsElements.size()-1).addEndPoint(new DotEndPoint(EndPointAnchor.BOTTOM));
    }

    metricsElements.get(metricsElements.size()-1).addEndPoint(new DotEndPoint(EndPointAnchor.TOP));
    if (allAspiceProcesses.stream().anyMatch(aspiceProcess -> aspiceProcess.getAspiceProcessId() == currMetric.getAsp
        metricsElements.get(metricsElements.size()-1).addEndPoint(new DotEndPoint(EndPointAnchor.TOP));
    }

    if (selectedMetrics != null) {
        if (selectedMetrics.contains(currMetric)) {
            metricsElements.get(metricsElements.size()-1).setStyleClass("ui-diagram-selected");
        }
    }
}

for (int i = 0; i < allMetricTypes.size(); i++) {
    MetricType currMetricType = allMetricTypes.get(i);

    metricTypesElements.add(new Element("Metric Type: " + currMetricType.getName(), String.valueOf(20*i)+"em", "36em"
    metricTypesElements.get(metricTypesElements.size()-1).addEndPoint(new DotEndPoint(EndPointAnchor.TOP));

    if (allMetrics.stream().anyMatch(metric -> metric.getMetricTypeId() == currMetricType.getMetricTypeId())) {
        metricTypesElements.get(metricTypesElements.size()-1).addEndPoint(new DotEndPoint(EndPointAnchor.TOP));

    }

    if (selectedMetricTypes != null) {
        if (selectedMetricTypes.contains(currMetricType)) {
            metricTypesElements.get(metricTypesElements.size()-1).setStyleClass("ui-diagram-selected");
        }
    }
}

for (int i = 0; i < allAspiceProcesses.size(); i++) {
    aspiceProcessesElements.add(new Element("AspiceProcess: " + allAspiceProcesses.get(i).getName(), String.valueOf(2
    aspiceProcessesElements.get(aspiceProcessesElements.size()-1).addEndPoint(new DotEndPoint(EndPointAnchor.BOTTOM))
    aspiceProcessesElements.get(aspiceProcessesElements.size()-1).addEndPoint(new DotEndPoint(EndPointAnchor.TOP));

    if (selectedAspiceProcesses != null) {
        if (selectedAspiceProcesses.contains(allAspiceProcesses.get(i))) {
            aspiceProcessesElements.get(aspiceProcessesElements.size()-1).setStyleClass("ui-diagram-selected");
        }
    }
}
```

Obrázek 23: Část zdrojového kódu pro zobrazení diagramu

3.9.2. Funkce pro zobrazení označených řádků v tabulce

Pro lepší uživatelský komfort jsem do všech tabulek přidal možnost zobrazení označených řádků uvnitř diagramu, kde se tyto řádky značí červenou barvou, včetně všech jejich propojení. Pro uživatele to znamená zlepšení přehlednosti databáze, zvlášť při velkém objemu zapsaných dat. Konkrétní implementace funguje tak, že na stránce každé tabulky jsem přidal tlačítko „View selected in Diagram“, kdy při kliknutí na toto tlačítko aplikace přesměruje uživatele na stránku diagramu a všechny označené řádky uvnitř diagramu vyznačí červeně. Vzhledem k tomu, že i propojení jsou značena

červenou barvou jsem dospěl k rozhodnutí, že pro tato propojení změním styl zobrazení po najetí myši na černou barvu.

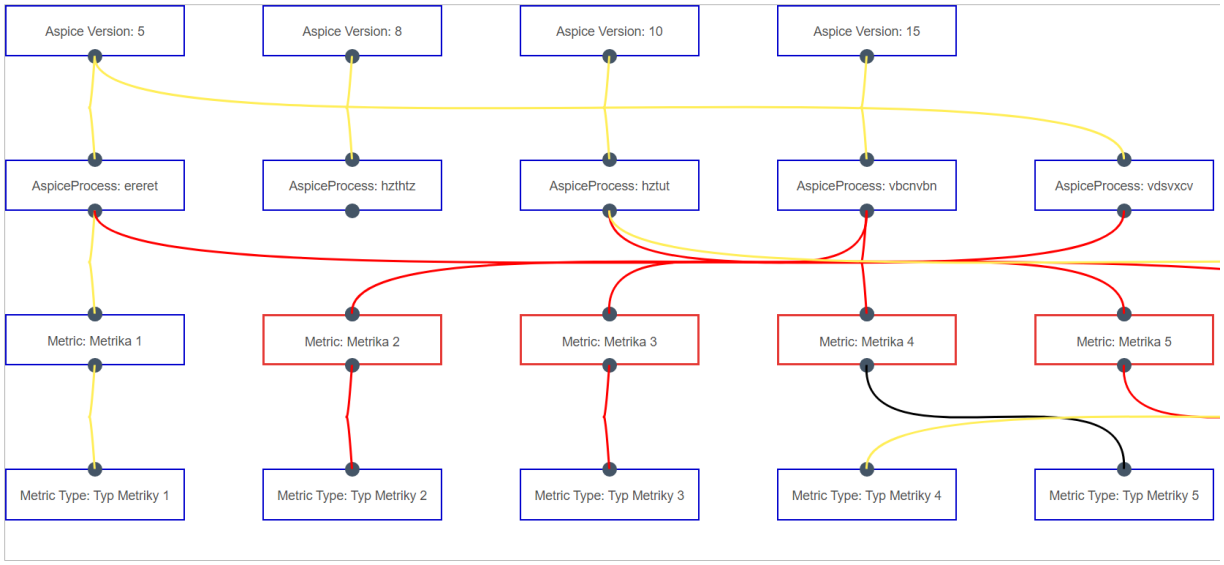
Metrics

Columns

Id	Name	Description	Properties	Severity	Satisfied	Data	Metric Type	Aspice Process	
7	Metrika 1	Popis Metriky 1	Vlastnosti Metriky 1	2	false	Data Metriky 1	Typ Metriky 1	ereret	
8	Metrika 2	Popis Metriky 2	Vlastnosti Metriky 2	3	true	Data Metriky 2	Typ Metriky 2	vdsvxcv	
9	Metrika 3	Popis Metriky 3	Vlastnosti Metriky 3	1	true	Data Metriky 3	Typ Metriky 3	vbcnvbn	
10	Metrika 4	Popis Metriky 4	Vlastnosti Metriky 4	3	true	Data Metriky 4	Typ Metriky 5	vbcnvbn	
11	Metrika 5	Popis Metriky 5	Vlastnosti Metriky 5	3	false	Data Metriky 5	Typ Metriky 6	hztut	
12	Metrika 6	Popis Metriky 6	Vlasnosti Metriky 6	1	true	Data Metriky 6	Typ Metriky 7	ereret	
13	Metrika 7	Popis Metriky 7	Vlastnosti Metriky 7	0	false	Data Metriky 7	Typ Metriky 4	hztut	

Add Metric | View selected in Diagram | Delete | PDF | CSV

Obrázek 24: Tabulka Metrics s novou funkcí „View selected in Diagram“



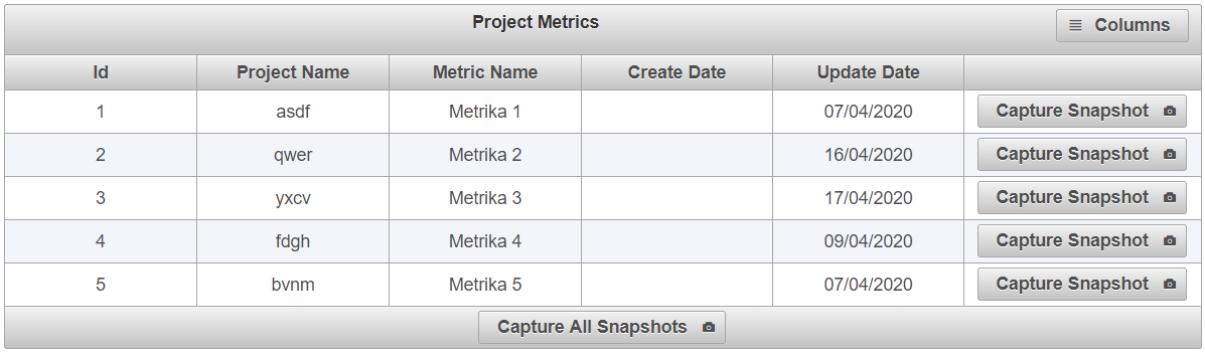
Obrázek 25: Prezentace vybraných řádků uvnitř diagramu

3.10. Stránka Project Metrics

Stránka Project Metrics slouží jako hlavní úložiště pro zachycení definic projektových metrik, jejich datumů vzniku a datumů poslední aktualizace. Zároveň jsem zde implementoval hlavní funkci pro dosažení cíle projektu, kterou je možnost zachycení aktuálních stavů projektových metrik. Díky zachycení stavů projektových metrik lze monitorovat vývoj kvality v čase.

3.10.1. Základní implementace tabulky

Základem bylo vytvořit rozhraní pro zobrazení dat uvnitř databáze. Tak samo, jako u ostatních tabulek jsem nadefinoval sloupce podle návrhu tabulek, ale s pár výjimkami. Oproti ostatním tabulkám jsem zde nepřidával možnost přidání, úpravu a mazání záznamů, protože tyto funkce patří do tabulky pro Projekty, která umožňuje uživatelům manipulaci s projektovými metrikami. Uvnitř databáze se do projektových metrik ukládá ID projektu a ID metriky, která s ní souvisí, uživateli se však stejně jako u ostatních tabulek obě tyto informace musí zobrazovat pod jejich jménem, ze kterého vyvodí více informací, než kdyby se zobrazovalo pouze ID.



Obrázek 26: Rozhraní pro zobrazení tabulky Project Metrics

3.10.2. Proměnlivý sloupeček „Update Date“

Každá projektová metrika má svou definovanou metriku. Při jakékoliv provedené změně této definované metriky je potřeba uvnitř projektové metriky tuto změnu zaznamenat. Jelikož může být jedna metrika součástí více projektových metrik, je potřeba, aby se sloupeček „Update Date“ (poslední aktualizace) při provedení změny uvnitř této metriky změnil na datum, kdy byla změna provedena u všech projektových metrik, které tuto metriku mají obsaženou. Toho jsem dosáhl tak, že jsem pozměnil kód pro editaci metrik tak, aby se detekovaly jakékoliv provedené změny a pokud tyto změny nastaly, získám veškeré projektové metriky, které ji mají obsaženou a aktualizuji výše zmíněny sloupeček na aktuální datum.

```
if (selectedMetric != null) { //edit
    if (metricExists(selectedMetric.getMetricId())) {

        boolean wasChange = !(selectedMetric.getName().equals(name) && selectedMetric.getDescription().equals(description)
                && selectedMetric.getProperties().equals(properties) && selectedMetric.getData().equals(data)
                && selectedMetric.getSeverity() == severity && (Boolean.compare(selectedMetric.getSatisfied(), satisfied) == 0 ? true
                && selectedMetric.getMetricTypeId() == getMetricTypeIdByName(metricTypeName)
                && selectedMetric.getAspiceProcessId() == getAspiceProcessIdByName(aspiceProcessName));

        selectedMetric.setName(name);
        selectedMetric.setDescription(description);
        selectedMetric.setProperties(properties);
        selectedMetric.setData(data);
        selectedMetric.setSeverity(severity);
        selectedMetric.setSatisfied(satisfied);
        selectedMetric.setMetricTypeId(getMetricTypeIdByName(metricTypeName));
        selectedMetric.setAspiceProcessId(getAspiceProcessIdByName(aspiceProcessName));
        MetricLocalServiceUtil.updateMetric(selectedMetric);

        if (wasChange) {
            List <ProjectMetric> projectMetricsWithThisMetric = ProjectMetricLocalServiceUtil.getAllProjectMetrics()
                    .stream().filter(projectMetric -> projectMetric.getMetricId() == selectedMetric.getMetricId()).collect(Collectors.

            projectMetricsWithThisMetric.forEach(projectMetric -> {
                projectMetric.setUpdateDate(new Date());
                ProjectMetricLocalServiceUtil.updateProjectMetric(projectMetric);
            });
        }
    }
```

Obrázek 27: Nový kód pro editaci metrik

3.10.3. Implementace zachycení aktuálního stavu projektové metriky

Pro dosažení zachycení aktuálního stavu projektové metriky bylo prvním krokem pro každý řádek přidat tlačítko „Capture Snapshot“ (zachycení snímku), které při kliknutí přidá do databáze záznamů projektových metrik aktuální záznam o projektové metrice, kterou chtěl uživatel zachytit. Vždy jsou zachycena všechna data, která se na danou projektovou metriku vztahují. Těmito daty jsou:

- datum zachycení záznamu
- název projektové metriky
- název metriky, která se na projektovou metriku vztahuje a její informace:
    - popis
    - data uvnitř
    - zda už byla metrika z pohledu zákazníka splněna
    - vlastnosti
    - náležitost
    - typ metriky
    - ASPICE Proces, který se na metriku vztahuje

Data se tedy extrahují z drtivé většiny databáze. Po domluvě se senior programátorem ve firmě jsme došli k závěru, že pro implementaci této extrakce dat bude pro její uložení uvnitř tabulky Project Metric Snapshots přidán sloupeček „value“ (hodnota), do kterého se tyto data budou ukládat a následně parsovat zpátky. Název value byl zvolen právě proto, protože záznamy uvnitř budou vázány vztahem typu key – value (klíč – hodnota), podobné datové struktuře mapa, v tomto případě tedy spíše multimapa, jelikož jeden klíč může obsahovat více hodnot. Jako klíčem v tomto případě může sloužit ID projektové metriky, která je ukládána, právě díky tomu, že hodnoty ID všech záznamů uvnitř databáze mají konstantní hodnotu, kterou uživatelé nevidí a pokud jim nedáme tu možnost (což by bylo nebezpečné, jelikož každý záznam musí mít unikátní ID), nemohou ID jakéhokoliv záznamu měnit.

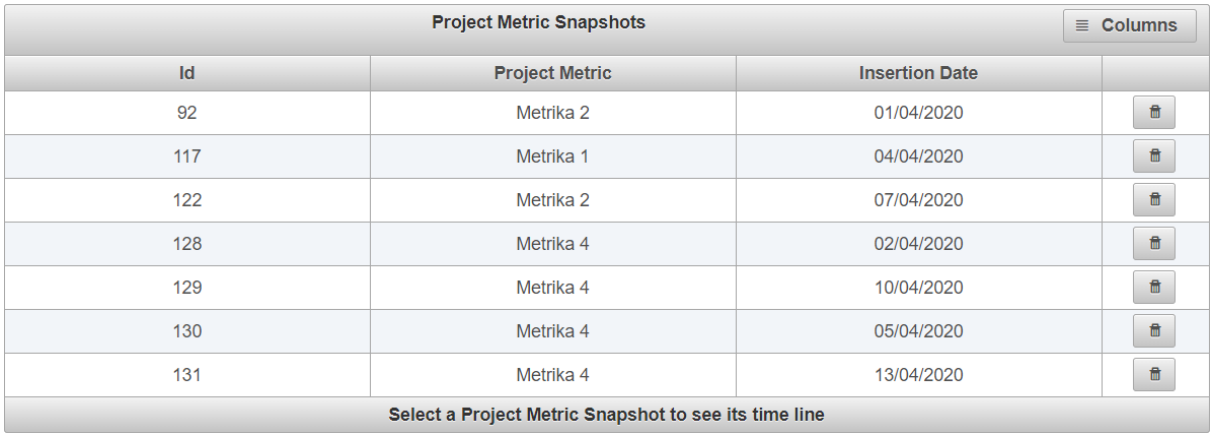
Po naplánování způsobu ukládání jsem tuto funkci pro zachycení snímku projektové metriky naprogramoval (obrázek 28), následně bylo potřeba vytvořit rozhraní pro zobrazení těchto snímků.

```
try {
    Metric metricFromProjectMetric = MetricLocalServiceUtil.
            getMetric(ProjectMetricLocalServiceUtil.getProjectMetric(selectedProjectMetric.getProjectMetricId()

    metricId = metricFromProjectMetric.getMetricId();
    metricNameWithId = metricFromProjectMetric.getName(); //+ " with ID: " + metricId;

    metricDescription = metricFromProjectMetric.getDescription();
    metricProperties = metricFromProjectMetric.getProperties();
    metricSeverity = metricFromProjectMetric.getSeverity();
    metricSatisfied = metricFromProjectMetric.getSatisfied();
    metricData = metricFromProjectMetric.getData();

    metricTypeId = metricFromProjectMetric.getMetricTypeId();
    metricTypeNameWithId = MetricTypeLocalServiceUtil.getMetricType(metricTypeId).getName(); //+ " with ID: " +

    aspiceProcessId = metricFromProjectMetric.getAspiceProcessId();
    aspiceProcessNameWithId = AspiceProcessLocalServiceUtil.getAspiceProcess(aspiceProcessId).getName(); //+ "

    Date currDate = new Date();
    String newValue = metricNameWithId + delimiter
                    + metricDescription + delimiter
                    + metricProperties + delimiter
                    + metricSeverity + delimiter
                    + metricSatisfied + delimiter
                    + metricData + delimiter
                    + metricTypeNameWithId + delimiter
                    + aspiceProcessNameWithId + delimiter
                    + currDate;

    ProjectMetricSnapshot metricSnapshot = ProjectMetricSnapshotLocalServiceUtil.createProjectMetricSnapshot(0)
    metricSnapshot.setValue(newValue);
    metricSnapshot.setInsertionDate(currDate);
    metricSnapshot.setProjectMetricId(selectedProjectMetric.getProjectMetricId());
    ProjectMetricSnapshotLocalServiceUtil.addProjectMetricSnapshot(metricSnapshot);
} catch (PortalException e) {
    e.printStackTrace();
}
```

Obrázek 28: Část kódu pro zachycení snímku projektových metrik

3.11. Stránka Project Metric Snapshots

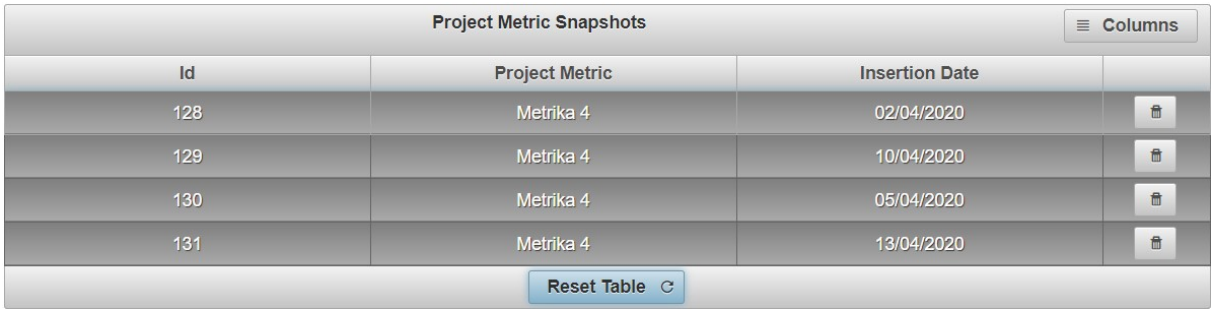
Prvním krokem stejně jako u všech stránek bylo zajistit rozhraní pro zobrazení tabulky. Opět není možno přidávat záznamy přímo z rozhraní této tabulky, jelikož jsem tuto funkci naprogramoval v předchozí tabulce. Jediné, co je zde povoleno je smazat jakýkoliv záznam z tabulky. Jakmile jsem toto rozhraní vytvořil, bylo potřeba zvolit vhodnou komponentu pro grafické zobrazení snímků v průběhu času, do které tyto data vkládat.



Obrázek 29: Zobrazení tabulky Project Metric Snapshots

3.11.1. Implementace grafického zobrazení v průběhu času

Pro zobrazení projektových metrik v čase jsem se rozhodl použít PrimeFaces komponentu „Timeline“ (časová linie). Prvním krokem pro toto zobrazení bylo vymyslet, jak se uživatel ke grafickému zobrazení dostane. Zvolil jsem přístup, kdy při kliknutí na jakýkoliv záznam snímku projektové metriky uvnitř tabulky se výběr projektových metrik automaticky zúží pouze na všechny záznamy projektové metriky, na kterou bylo kliknuto a zobrazí se časová linie této projektové metriky. Jelikož zúžím záznamy projektových metrik, je důležité, aby tento seznam mohl uživatel zpátky resetovat na zobrazení všech záznamů. Proto jsem se rozhodl přidat tlačítko „Reset Table“ (resetování tabulky), které se zobrazuje pouze tehdy, kdy má uživatel zúžený výběr záznamů.



Obrázek 30: Zobrazení po kliknutí na projektovou metriku

3.11.2. Implementace komponenty časové linie (Timeline)

Umístění časové linie jsem určil přímo pod tabulku vybraných záznamů. Po prostudování dokumentace komponenty „Timeline“ jsem zjistil, že pro tvoření nových událostí uvnitř časové linie existuje možnost předat jako parametr pro tvoření dat objekty z vlastní třídy, pokud tato třída splňuje určitý formát. Ke každé této události je i potřeba určit počáteční datum, které musí být specifikováno proměnnou typu „LocalDateTime“. Začal jsem tedy tím, že jsem vytvořil vlastní třídu, na kterou se vztahují objekty, které budu předávat jako data pro tvoření událostí a zároveň i samostatnou třídu, kterou

jsem pojmenoval „TimelineView“, uvnitř které jsem programoval veškerou logiku pro tvoření časové linie a dat uvnitř.

```
private String metricNameWithId;
  private String metricDescription;
  private String metricProperties;
  private String metricSeverity;
  private String metricSatisfied;
  private String metricData;
  private String metricTypeNameWithId;
  private String aspiceProcessNameWithId;
  private String insertionDate;

  public CustomSnapshotObject(String metricNameWithId, String metricDescription, String metricProperties, String metricSeverity
         String metricSatisfied, String metricData, String metricTypeNameWithId, String aspiceProcessNameWithId, String inserti
      this.metricNameWithId = /*"<strong>Metric: </strong>" + */ metricNameWithId;
      this.metricDescription = /*"<strong>Description: </strong>" + */ metricDescription.replace("<p>", "").replace("</p>", "")
      this.metricProperties = /*"<strong>Properties: </strong>" + */ metricProperties;
      this.metricSeverity = /*"<strong>Severity: </strong>" + */ metricSeverity;
      this.metricSatisfied = /*"<strong>Satisfied: </strong>" + */ metricSatisfied;
      this.metricData = /*"<strong>Data: </strong>" + */ metricData;
      this.metricTypeNameWithId = /*"<strong>Metric Type: </strong>" + */ metricTypeNameWithId;
      this.aspiceProcessNameWithId = /*"<strong>Aspice Process: </strong>" + */ aspiceProcessNameWithId;
      this.insertionDate = /*"<strong>Inserted on: </strong>" + */ insertionDate;
  }

//GETTERY, SETTERY... ↓
```

Obrázek 31: Část kódu vlastní třídy pro zapouzdření dat o události

Dále bylo potřeba zpětně parsovat zaznamenaná data ze sloupce „value“ každého záznamů snímku projektové metriky, na který se vztahoval výběr uživatele. Jelikož je u každé události i zapotřebí specifikovat proměnnou typu „LocalDateTime“, musel jsem každý záznam specifikovat jako pár, kde první hodnotou bylo pole řetězců, které obsahovaly parsované data a druhou hodnotou proměnná typu „Date“. Vzhledem k tomu, že jazyk Java neobsahuje implementaci pro tvoření párů různých typů, vytvořil jsem svou vlastní třídu, která sloužila mým účelům pro tvoření těchto párů. Jakmile jsem toto dokončil, dalším krokem bylo naimplementovat logiku přidávání těchto záznamů jako události do časové linie. Využil jsem hodnoty z pole řetězců, které jsem ukládal jako první hodnotu páru a tvořil jsem z nich objekt, který sloužil jako kontejner dat pro události. Pro určení počátečního data jsem vytvořil metodu, která přijímala vstupní parametr typu „Date“, který následně převedla na požadovaný typ „LocalDateTime“ a ten jako návratovou hodnotu vrátila. Metodu jsem vždy použil pro převod druhé hodnoty páru typu „Date“ na určení počátečního data události typu „LocalDateTime“.

```
model = new TimelineModel<>();

long projectMetricIdFromClickedSnapshot = clickedSnapshot.getProjectMetricId();
List <ProjectMetricSnapshot> allProjectMetricSnapshots = ProjectMetricSnapshotLocalServiceUtil.getAllProjectMetricSnapshots();
List <ProjectMetricSnapshot> snapshotsWithClickedMetricId = allProjectMetricSnapshots
        .stream().filter(snapshot -> snapshot.getProjectMetricId() == projectMetricIdFromClickedSnapshot).collect(Collectors.t

List <SnapshotPair> valuesFromSelectedSnapshots = new ArrayList<SnapshotPair>();
snapshotsWithClickedMetricId.forEach(snapshot ->
        valuesFromSelectedSnapshots.add(new SnapshotPair(snapshot.getValue().split(ProjectMetricView.delimiter), snapshot.getI

/* ---PARSOVANI---
 * [0] metricNameWithId
 * [1] metricDescription
 * [2] metricProperties
 * [3] metricSeverity
 * [4] metricSatisfied
 * [5] metricData
 * [6] metricTypeNameWithId
 * [7] aspiceProcessNameWithId
 */

for (int i = 0; i<valuesFromSelectedSnapshots.size(); i++) {

    SnapshotPair sp = valuesFromSelectedSnapshots.get(i);
    DateFormat dateformat = new SimpleDateFormat("dd-MM-yyyy HH:mm:ss");

    if (i != valuesFromSelectedSnapshots.size()-1) {
        model.add(TimelineEvent.<Object>builder().data(new CustomSnapshotObject(
                sp.getValues()[0],
                sp.getValues()[1],
                sp.getValues()[2],
                sp.getValues()[3],
                sp.getValues()[4],
                sp.getValues()[5],
                sp.getValues()[6],
                sp.getValues()[7],
                dateformat.format(sp.getInsertionDate())))
                .startDate(convertToLocalDateTime(sp.getInsertionDate()))
                .endDate(convertToLocalDateTime(valuesFromSelectedSnapshots.get(i+1).getInsertionDate()))
                .styleClass((sp.getValues()[4].equals("true") ? "satisfied" : "unsatisfied"))
                .build());
```

Obrázek 32: Část kódu pro přidávání záznamů jako události uvnitř časové linie

Dále jsem uvnitř xhtml souboru stránky Project Metric Snapshots definoval parametry a styly pro zobrazení časové linie a způsob zobrazení dat uvnitř událostí. Hodnoty parametrů jsem nadefinoval uvnitř třídy „TimelineView“ a přes její Java Bean, který jsem pojmenoval „timelineView“ jsem tyto hodnoty získával.

Mezi tyto parametry a mé nastavení se řadilo:

- přiblížení/oddálení časové linie – povoleno
- zobrazení datumů na spodku časové linie – povoleno
- zda lze s různými událostmi po časové linii pohybovat – nepovoleno
- zda se má zobrazovat vertikální červená osa, která značí dnešní datum – povoleno
- styl zobrazení události – čtverec
- při překrývaní dvou či více událostí tyto události poskládat nad sebe – povoleno
- styl časové osy – definoval jsem vlastní styl „timeline-axis“, kde barva časové osy je černá a šířka okraje je 5 pixelů
- zpoždění zobrazení nápovědy (popisek, který se zobrazí při najetí myši na událost) – 0 milisekund
- zda má nápověda následovat myš – povoleno
- umístění nápovědy vůči myši – roh popisku vždy u myši

Způsob zobrazení dat uvnitř události jsem vzhledem k tomu, že každá událost má mnoho různých řádků, které mají na základě zobrazovaného snímků projektové metriky proměnlivou velikost, rozdělil na dvě části:

- První část, která se zobrazuje vždy: do této části jsem umístil informace, které by měly mít buď konstantní délku, nebo délku, která se mění jen minimálně. Důvodem je, že pokud by se na každé události zobrazovaly kvanta informací, vypadalo by to graficky nevzhledně.
- Druhá část, která se zobrazuje jen tehdy, kdy uživatel přes událost přejede myší (nápověda). Uvnitř této části se zobrazují informace, které mohou mít velký objem informací, jako třeba data a popis metriky, nebo informace, které nejsou až tak potřebné, jako je upřesnění datumu vložení záznamu na hodiny, minuty a sekundy.

Obě části čerpají data z objektu třídy, kterou jsem vytvořil pro ukládání dat události.

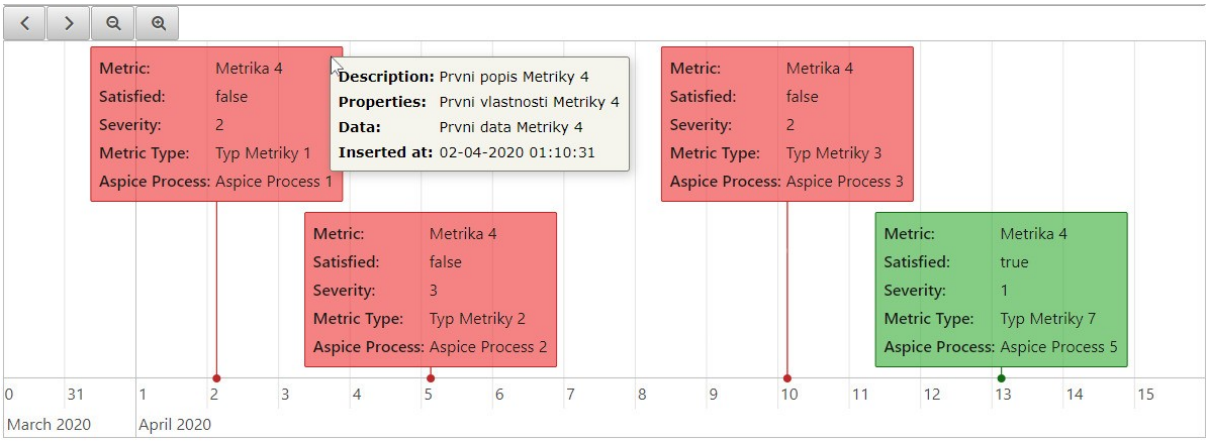
```
<p:timeline id="snapshotTimeline" value="#{timelineView.model}" var="snapshot" rendered="#{projectMetricSnapshotView.showT
selectable="#{timelineView.selectable}" tooltipFollowMouse="true" tooltipDelay="0" tooltipOverflowMethod="flip"
zoomable="#{timelineView.zoomable}" widgetVar="timelineWdgt"
moveable="#{timelineView.moveable}" eventStyle="#{timelineView.eventStyle}" stackEvents="#{timelineView.stackEvents}"
axisOnTop="#{timelineView.axisOnTop}" styleClass="timeline-axis"
showCurrentTime="#{timelineView.showCurrentTime}">
<p:ajax event="select" listener="#{timelineView.onSelect}" update=":projectMetricSnapshotForm:projectMetricSnapshotMsgs"/>

<h:panelGrid columns="2" style="text-align:left;">
    <strong>Metric:</strong>
    <h:outputText style="text-align:left;" escape="false" value="#{snapshot.metricNameWithId}"/>

    <strong>Satisfied:</strong>
    <h:outputText style="text-align:left;" escape="false" value="#{snapshot.metricSatisfied}"/>

    <strong>Severity:</strong>
    <h:outputText style="text-align:left;" escape="false" value="#{snapshot.metricSeverity}" />

    <strong>Metric Type:</strong>
    <h:outputText style="text-align:left;" escape="false" value="#{snapshot.metricTypeNameWithId}"/>

    <strong>Aspice Process:</strong>
    <h:outputText style="text-align:left;" escape="false" value="#{snapshot.aspiceProcessNameWithId}"/>
</h:panelGrid>
<f:facet name="eventTitle">
    <h:panelGrid columns="2" style="text-align:left;">
        <strong>Description:</strong>
        <h:outputText escape="false" value="#{snapshot.metricDescription}" />

        <strong>Properties:</strong>
        <h:outputText escape="false" value="#{snapshot.metricProperties}" />

        <strong>Data:</strong>
        <h:outputText escape="false" value="#{snapshot.metricData}" />

        <strong>Inserted at:</strong>
        <h:outputText escape="false" value="#{snapshot.insertionDate}" />
    </h:panelGrid>
</f:facet>
```

Obrázek 33: Zdrojový kód pro časovou linii a způsob zobrazení dat uvnitř

Při studování dokumentace pro komponentu časové linie jsem se dozvěděl, že pro jednotlivé události lze tvořit vlastní CSS styly, což mi přišlo vhod pro obohacení uživatelské zkušenosti. Posledním krokem tedy bylo definovaní dvou CSS stylů, které závisely na tom, zda byla daná metrika z pohledu zákazníka splněna. Pokud byla splněna, aplikuje se styl s odstínem zelené barvy, pokud nebyla, aplikuje se ten samý styl s odstínem červené barvy.

Na obrázku 34 je zobrazen můj uměle vytvořený příklad tohoto vývoje, kde lze na časové linii vidět vývoj jedné projektové metriky a pozorovat změny provedené uvnitř. Projektová metrika je na

každém snímku z pohledu zákazníka označena jako nesplněna, až se nakonec tato hodnota změní a aplikuje se i jiný styl.

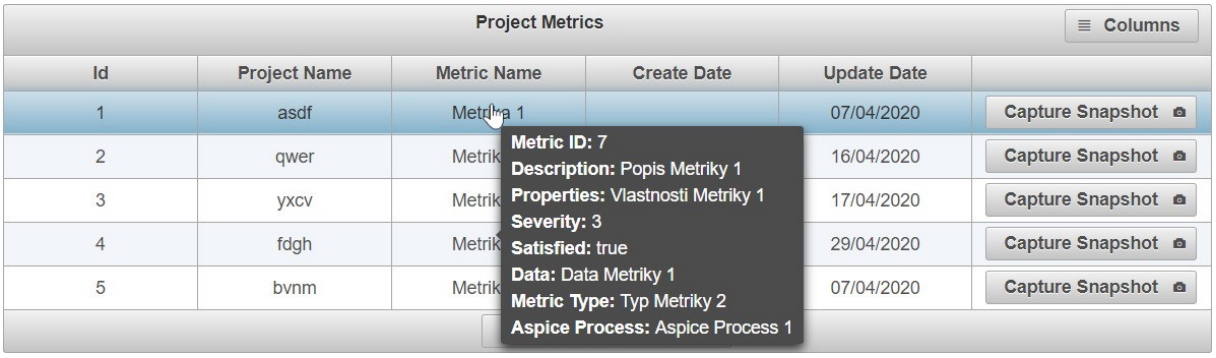


Obrázek 34: Vizualizace snímků projektových metrik na časové linii se znázorněnou nápovědou

3.12. Nápověda u tabulek

Dalším úkolem bylo naimplementovat nápovědy i do ostatních tabulek v databázi. Níže je uvedený seznam tabulek a sloupce, pod kterými by se měly uživateli zobrazovat upřesňující informace o jakémkoliv záznamu, na který najede myší:

- Tabulka Project Metrics:
  - U jména projektu zobrazit jeho ID a popis
  - U jména metriky její ID, popis, vlastnosti, důležitost, zda byla splněna, data, její typ a ASPICE proces, na který se vztahuje
- Tabulka Project Metric Snapshots:
  - U projektové metriky zobrazit jméno projektu, na který se vztahuje, ID metriky, datum vytvoření a poslední datum aktualizace
- Tabulka Metrics:
  - U sloupce typu metriky zobrazit jeho ID, konkrétní typ a popis
  - Pod ASPICE procesem zobrazit jeho ID, jméno, popis, číslo verze a ID této verze
- Tabulka ASPICE Processes: pod sloupcem ASPICE version zobrazit ID verze, její popis a datum vydání



Obrázek 35: Ukázka nápovědy u tabulky Project Metrics

Pro implementaci těchto nápověd bylo jako prvním krokem potřeba u všech sloupců, na které se nápovědy vztahovaly, definovat PrimeFaces komponentu „p:tooltip“ a určit parametry pro tuto nápovědu. Parametry jsem vždy nastavil tak, aby se nápověda zobrazila napravo od myši a pozice se dynamicky měnila na základě pozice myši. Dále jsem vždy nastavil zpoždění zobrazení nápovědy na 0 milisekund a pokud je uvnitř nápovědy HTML značka, například když je text vnořený mezi štítkem „“ a jeho ukončujícím štítkem, tak se tento text bude zobrazovat tučně a štítek „“ nepůjde vidět. Dále bylo potřeba vždy definovat zdroj, z kterého tato nápověda bude čerpat informace, které se následně zobrazí uživateli. Pro každou nápovědu jsem vždy na základě toho, uvnitř které tabulky se nacházel, vytvořil uvnitř Java Beanu pro tuto tabulku metodu na získání těchto informací a následně tuto metodu odkázal jako zdroj informací pro danou nápovědu.

```
try {
    Metric specificMetric = MetricLocalServiceUtil.getMetric(number);
    String metricTypeName = MetricTypeLocalServiceUtil.getMetricType(specificMetric.getMetricId()).getName();
    String aspiceProcessName = AspiceProcessLocalServiceUtil.getAspiceProcess(specificMetric.getAspiceProcessId()).getName();

    return "<strong>Metric ID: </strong>" + specificMetric.getMetricId() + "<br/>"
           + (specificMetric.getDescription() != null ? "<strong>Description: </strong>" + specificMetric.getDescription().replace("<p>"
           + (specificMetric.getProperties() != null ? "<strong>Properties: </strong>" + specificMetric.getProperties() + "<br/>" : "")
           + (specificMetric.getSeverity() != -1 ? "<strong>Severity: </strong>" + specificMetric.getSeverity() + "<br/>" : "")
           + "<strong>Satisfied: </strong>" + specificMetric.isSatisfied() + "<br/>"
           + (specificMetric.getData() != null ? "<strong>Data: </strong>" + specificMetric.getData() + "<br/>" : "")
           + (metricTypeName != null ? "<strong>Metric Type: </strong>" + metricTypeName + "<br/>" : "")
           + (aspiceProcessName != null ? "<strong>Aspice Process: </strong>" + aspiceProcessName + "<br/>" : "");

} catch (Exception ex) {
    System.out.println(ex);
    return "";
}
```

Obrázek 36: Příklad zdrojového kódu pro získání informací pro nápovědy

```
<p:column headerText="Metric Name" width="6%" style="text-align: center;">
        <h:outputText id="metricTrack" value="#{projectMetricView.getMetricName(projectMetricTable.metricId)}" />
        <p:tooltip id="metricTooltip" for="metricTrack" position="right" showDelay="0" trackMouse="true"
            escape="false" value="#{projectMetricView.getMetricValues(projectMetricTable.metricId)}">
        </p:tooltip>
</p:column>
```

Obrázek 37: Příklad zdrojového kódu pro definici nápovědy

3.13. Vylepšení implementace pro zobrazení snímků projektových metrik

Předchozí implementace zobrazení časové linie fungovala pouze tehdy, pokud byl uživatel na stránce Project Metric Snapshots, kde se nacházela tabulka pro snímky projektových metrik a uvnitř této tabulky kliknul na snímek. Vzhledem k tomu, že se tyto snímky vždy vážou na určité projektové metriky, bylo vhodné, aby byla tato časová linie dostupná oboustranně, kdy uživatel na stránce Project Metrics uvnitř tabulky klikne na projektovou metriku a bude přesměrován na zobrazení časové linie všech uložených snímků této projektové metriky, pokud nějaké existují.

Začal jsem tím, že jsem pozměnil kód pro tvoření časové linie tak, aby správně fungoval i když bude volán z jiné stránky.

Dále jsem vytvořil metodu uvnitř Java Beanu pro stránku Project Metrics, která se zavolá kdykoliv uživatel klikne na řádek uvnitř tabulky. Bylo potřeba i zajistit kontrolu, zda je projektová metrika, na kterou uživatel kliknul přítomná v tabulce se snímky a jestli je, přesměrovat uživatele na zobrazení časové linie této projektové metriky.

```
selectedProjectMetric = event.getObject();
try {
    Metric metricOfThisProject = MetricLocalServiceUtil.getMetric(selectedProjectMetric.getMetricId());
    Optional <ProjectMetricSnapshot> projectMetricSnapshotWithThisMetric = ProjectMetricSnapshotLocalServiceUtil.getAllProjectMetricSnaps
            .stream().filter(snapshot -> {
                try {
                    return ProjectMetricLocalServiceUtil.getProjectMetric(snapshot.getProjectMetricId()).getMetricId() == metricOfThisPro
                } catch (PortalException e) {
                    System.out.println(e);
                }
                return false;
            }).findFirst();

    if (projectMetricSnapshotWithThisMetric.isPresent()) {
        projectMetricSnapshotView.CreateTimeline(projectMetricSnapshotWithThisMetric.get());
        FacesContext.getCurrentInstance().getApplication().getNavigationHandler().
        handleNavigation(FacesContext.getCurrentInstance(), null, "viewProjectMetricSnapshot.xhtml");
        projectMetricSnapshotView.setWasRedirectedFromProjectMetric(true);
    }
} catch (PortalException e) {
    System.out.println(e);
```

Obrázek 38: Část zdrojového kódu pro zobrazení časové linie projektové metriky

4. TEORETICKÉ A PRAKTICKÉ ZNALOSTI A DOVEDNOSTI ZÍSKANÉ V PRŮBĚHU PRAXE

Z hlediska nabytých znalostí a dovedností mi absolvování této odborné praxe převýšilo očekávání. Získal jsem porozumění o tom, co vše je potřeba k vývoji aplikace, která využívá práci s databází a jejím rozhraním pro interakci s uživatelem je webová stránka. Získané znalosti a dovednosti bych shrnul takto:

- První dovedností, kterou bych začal je určitě zlepšení zdatností v kódování. Jak plynul čas, všimnul jsem si, že řešení různých problémů mi dělá čím dál tím menší potíže, včetně i samotné implementace řešení, která mi trvala čím dál tím kratší dobu. Nemůžu opomenout i prohloubení porozumění jazyku Java, zejména práce s Javovskými lambda výrazy, které jsem používal hlavně při práci se seznamy.
- Zprovoznění, konfigurace, administrace webového serveru a propojení s databází pomocí Liferay Portal a Apache Tomcat.
- Generování uživatelského rozhraní a logiky uvnitř pomocí JSF. Tahle technologie pro mě byla nejtěžší na porozumění a na začátku mi i přes prostudování dělala potíže. Hlavním důvodem bylo, že zde například oproti běžnému programovacímu jazyku neexistuje žádný sofistikovanější kompilátor, který důkladně kontroluje syntaxi a podobné problémy, tím pádem jsem se často dozvěděl chyby v kódu teprve až při vyskočení běhové chyby, které i tak občas nebyly moc nápomocné pro ladění programu a bývaly o mnoho zákeřnější než ty, se kterými se setkávám u klasických programovacích jazyků. Postupem času jsem si principy práce s JSF osvojil a ke konci, například při tvorbě rozhraní a logiky pro stránku Project Metric Snapshots jsem už měl jen minimální potíže s implementací i složitějších operací.
- Práce s rámcem pro tvorbu uživatelského rozhraní. Díky práce s PrimeFaces jsem měl možnost si vyzkoušet, jak využít předpřipravené kódy pro komponenty a jak tyto komponenty na základě dokumentace, která je dostupná skoro u každé komponenty, přizpůsobit pro vlastní potřeby, například vlastní styly grafů a časových linií.
- Správa databáze pomocí DBeaveru. Pro jakýkoliv projekt, který pracuje s databází je potřeba umět rychlou a jednoduchou manipulací s danou databází. Pomocí DBeaveru jsem se naučil vytvořit spojení s databází a měnit uvnitř vlastnosti sloupců, přidávat/upravovat/mazat testovací data, tvořit/aktualizovat/měnit nové tabulky a kontrolovat správnost dat, které pocházely jako výstup z programu.
- Komunikace a práce s databází pomocí Service Builderu. Má zkušenost s komunikaci s databází pomocí Service Builderu se velmi liší od té typické, která se většinou skládá z řetězení SQL dotazů, jelikož při vytváření tabulek přes Service Builder se vytváří i modely těchto tabulek v podobě Java tříd, přes které se následně komunikuje s databázi. Naučil jsem se tyto třídy i přizpůsobovat pro vlastní potřeby komunikace s databázi.
- Práce v týmu a s verzovacím systémem. Jelikož jsem na projektu pracoval s dalším studentem, získal jsem i cenné zkušenosti pro práci v týmu. I když jsme každý pracovali nezávisle na sebe, sdíleli jsme mezi sebou nápady a na základě toho plánovali další kroky. Dále jsem se naučil práci s Gitem a jeho důležitost při práci v týmu, zejména pro zpětnou vazbu na kód od senior programátora.

5. ZÁVĚR

V první části bakalářské práce jsem sumarizoval technologie a nástroje, které sloužily jako prostředek pro tvorbu aplikace a ke každé shrnul její přínosy k projektu.

V druhé části jsem popsal průběh realizace projektu od začátku do konce a rozvedl řešení úkolů, které mi byly zadávány.

Výsledkem byla úspěšně navržena a naimplementována databáze pro ukládaní metrik a vyvinut systém, který zajistil webové rozhraní pro uživatele, díky kterému lze manipulovat s projektovými metrikami a sledovat jejich vývoj v čase. Na projektové metriky lze napojit konkrétní metriky, u kterých byla naimplementována možnost plné manipulace v podobě přidávání, úprav, mazání a napojení na proces, ke kterému se vztahují. Toto platí i pro všechny další části systému, na které se vztahuje možnost přizpůsobení dle vlastních potřeb. Pro projektové metriky byla i naimplementována vlastnost zachycení aktuálního snímků a následné zobrazení všech snímků této projektové metriky na časové linii, což umožňuje sledování změn vlastností metrik v čase. Uživatel má i možnost zobrazit diagram celého systému, který znázorňuje vztahy a propojení entit uvnitř databáze, včetně možnosti zvýraznění volitelného selektivního výběru určitých části.

Celkovou zkušenost a přínos absolvování odborné praxe hodnotím velmi kladně. Práce mě bavila a byla to pro mě cenná příležitost, díky které jsem porozuměl různým fázím vývoje aplikace a naučil jsem se spoustu nových věcí, které mohu v budoucnu uplatnit v kariéře.

LITERATURA

[1] SCOVECO s.r.o. [online]. [cit. 2020-03-10]. Dostupné z: https://www.scoveco.cz/cs

[2] Java [online]. [cit. 2020-03-10]. Dostupné z: https://java.com/en/download/faq/whatis_java.xml

[3] Interpret [online]. [cit. 2020-03-10]. Dostupné z: http://staffwww.fullcoll.edu/nniccolai/Interpreter_Defn.htm

[4] Java jako programovací jazyk [online]. [cit. 2020-03-10]. Dostupné z: https://www.oracle.com/technetwork/java/intro-141325.html#367

[5] Java API [online]. [cit. 2020-03-11]. Dostupné z: https://docs.oracle.com/javase/tutorial/getStarted/intro/cando.html

[6] Oracle Jakarta EE [online]. [cit. 2020-03-11]. Dostupné z: https://docs.oracle.com/javaee/6/firstcup/doc/gkhoy.html

[7] Javatpoint Jakarta EE [online]. [cit. 2020-03-11]. Dostupné z: https://www.javatpoint.com/java-ee

[8] HTTP Cookie [online]. [cit. 2020-03-13]. Dostupné z: https://networkencyclopedia.com/http-cookie/#what-is-html-cookie

[9] Theserverside Jakarta EE [online]. [cit. 2020-03-13]. Dostupné z: https://www.theserverside.com/definition/J2EE-Java-2-Platform-Enterprise-Edition

[10] JSON [online]. [cit. 2020-03-13]. Dostupné z: https://www.json.org/json-en.html

[11] XML [online]. [cit. 2020-03-13]. Dostupné z: https://www.w3schools.com/xml/xml_whatis.asp

[12] Vzdálené volání procedur [online]. [cit. 2020-03-13]. Dostupné z: https://web.cs.wpi.edu/~cs4514/b98/week8-rpc/week8-rpc.html

[13] Eclipse IDE [online]. [cit. 2020-03-14]. Dostupné z: https://www.eclipse.org/eclipseide

[14] Eclipse IDE [online]. [cit. 2020-03-14]. Dostupné z: https://www.eclipse.org/ide

[15] Eclipse Tržiště [online]. [cit. 2020-03-14]. Dostupné z: https://marketplace.eclipse.org

[16] Eclipse Distribuce [online]. [cit. 2020-03-14]. Dostupné z: https://www.eclipse.org/downloads/packages

[17] Gradle úvod [online]. [cit. 2020-03-15]. Dostupné z: https://docs.gradle.org/current/userguide/userguide.html#new_projects_with_gradle

[18] Gradle firmy [online]. [cit. 2020-03-15]. Dostupné z: https://gradle.org

[19] Android Studio [online]. [cit. 2020-03-15]. Dostupné z: https://developer.android.com/studio/intro

[20] Gradle hlavní rysy [online]. [cit. 2020-03-15]. Dostupné z: https://gradle.org/maven-vs-gradle

[21] Gradle rysy [online]. [cit. 2020-03-15]. Dostupné z: https://gradle.org/features

[22] JavaServer Faces [online]. [cit. 2020-03-15]. Dostupné z: http://www.javaserverfaces.org

[23] DOBNIK Leon a ŠALEJ Marcel. Developing Rich Web Applications with PrimeFaces in Java EE7 [online], 1-5 [cit. 2020-03-16]. Dostupné z: https://drive.google.com/file/d/0B7kPrfCRft_qZzZyeUhmWlVXZU0/preview?pli=1

[24] JAMES GARRETT, Jesse. Ajax: A New Approach to Web Applications [online]. 2005, 1 [cit. 2020-03-16]. Dostupné z: https://courses.cs.washington.edu/courses/cse490h/07sp/readings/ajax_adaptive_path.pdf

[25] Fortune 500 [online]. [cit. 2020-03-16]. Dostupné z: https://fortune.com/fortune500

[26] Primefaces [online]. [cit. 2020-03-16]. Dostupné z: https://www.primefaces.org

[27] Liferay Portal [online]. [cit. 2020-03-16]. Dostupné z: https://portal.liferay.dev/discover/architecture

[28] Liferay Portlet [online]. [cit. 2020-03-16]. Dostupné z: https://portal.liferay.dev/docs/7-0/tutorials/-/knowledge_base/t/portlets

[29] Service Builder [online]. [cit. 2020-03-16]. Dostupné z: https://portal.liferay.dev/docs/7-2/appdev/-/knowledge_base/a/service-builder#service-builder

[30] Partneři Liferay [online]. [cit. 2020-03-16]. Dostupné z: https://www.liferay.com/services/partners

[31] JDBC Ovladač [online]. [cit. 2020-03-17]. Dostupné z: https://docs.oracle.com/javase/tutorial/jdbc/overview/index.html

[32] DBeaver [online]. [cit. 2020-03-17]. Dostupné z: https://dbeaver.io/about

[33] DBeaver [online]. [cit. 2020-03-17]. Dostupné z: https://github.com/dbeaver/dbeaver/wiki

[34] MariaDB [online]. [cit. 2020-03-17]. Dostupné z: https://mariadb.org

[35] REUTER, Andreas a HAERDER Theo. ACID. *Principles of Transaction-Oriented Database Recovery* [online]. 1983, 4-6 [cit. 2020-03-17]. Dostupné z: https://sites.fas.harvard.edu/~cs265/papers/haerder-1983.pdf

[36] Srovnání MariaDB a MySQL [online]. [cit. 2020-03-17]. Dostupné z: https://mariadb.com/kb/en/mariadb-vs-mysql-features/

[37] Microsoft Azure [online]. [cit. 2020-03-17]. Dostupné z: https://docs.microsoft.com/en-us/azure/mariadb

[38] Alibaba Cloud [online]. [cit. 2020-03-17]. Dostupné z: https://www.alibabacloud.com/products/apsaradb-for-rds-mariadb

[39] Amazon RDS [online]. [cit. 2020-03-17]. Dostupné z: https://aws.amazon.com/rds/mariadb

[40] Firmy využívající MariaDB [online]. [cit. 2020-03-17]. Dostupné z: https://mariadb.com/resources/customer-stories/

[41] Java Servlet [online]. [cit. 2020-03-19]. Dostupné z: http://www.cs.virginia.edu/~up3f/cs4501/supplement/basic-servlet.html

[42] Apache Tomcat [online]. [cit. 2020-03-19]. Dostupné z: http://tomcat.apache.org

[43] Napojení na Tomcat [online]. [cit. 2020-03-19]. Dostupné z: https://tomcat.apache.org/tomcat-4.1-doc/RUNNING.txt

[44] Git [online]. [cit. 2020-03-23]. Dostupné z: https://git-scm.com/

[45] O Gitu [online]. [cit. 2020-03-23]. Dostupné z: https://git-scm.com/about/branching-and-merging

[46] Automotive SPICE [online]. [cit. 2020-03-25]. Dostupné z: http://www.automotivespice.com/fileadmin/software-download/AutomotiveSPICE_PAM_31.pdf